Autonomous Aerial System

Jerrin bright
Vellore Institute of Technology, Chennai
School of Mechanical Engineering

Objective: Making an autonomous drone capable of navigating autonomously both in indoor and outdoor environments in a robust manner. Breaking it into sub-parts, pose estimation and mapping has been split amongst two teams to be made from scratch considering the environmental conditions (outdoor, indoor). Simulating the autonomous drone algorithm developed by the team was aimed to be tested in ROS platform using Gazebo Simulator.

Abstract: SLAM; Localization; Drones; ROS

Introduction

Drone: A drone is a flying robot that can be remotely controlled or fly autonomously.

UAV and Drones are interchangeably used. But UAVs are usually autonomous vehicles, whereas flying robots which are also controlled remotely through humans are termed as drones.

UGV are unmanned ground vehicles. These robots operate without human presence on the robot, it may be controlled remotely or maybe autonomous.
UAS are unmanned aircraft systems. This includes everything from the robot that flies to the person controlling it. This connects the person controlling it and the aerial robot. UAV are a subset of UAS.

Drone technology is continuously evolving. Some of the most common and widely used drone technologies are:

Radar positioning: With the help of GNSS, drones are controlled more efficiently and accurately. It accurately shows the position of the drone wrt to the controller. Obstacle avoiding and collision avoid ing technology: With the help of sensors and SLAM technology, the surroundings of the drone is accurately mapped and using software algorithms, the obstacles or collisions are avoiding.

Drone stability: The gyroscope embedded in drones helps it to stabilize its flight and provides navigational data.

Motors and propellers: These are used to make the drone move, fly and perform maneuvers.
Cameras: These are used in drones for videography and other purposes. The various types of drones are:

Multi rotor drones: These drones give great control and are cheap. They aren't very efficient as a lot of energy is used to fight gravity and hence their flight time is comparatively less and reduces as payload weight increases.
Fixed wing: These use wings for getting the lift. These are more efficient as they fight only air resistance and not gravity directly. Therefore, their flight duration is also long. But the maneuverability from fixed wing drones is less. They cannot vertically takeoff or land.
Single rotor: This has one main rotor for lift and one tail rotor to change direction. They offer better control and are more efficient because of the long blades. They can take off and land vertically and can also hover. It requires more maintenance because of its mechanical complexity.

Drones are used in a variety of domains. Some of them are:

- Aerial videography: Because of their compact size, and stability like in multi rotor drones, they are preferred for aerial videography and live coverage of sports.
- Shipping: These are also used for shipping and delivery.
- Geographic mapping: With the help of drones, mapping is more accurate and faster.
- Disaster management: Because of their small size, they can be easily deployed in various locations to get viable data about the situation in that area.
- Military: Drones are used in military as they reduce the risk because they are unmanned.
- Agriculture: With their speed and technologies, they are used to monitor crop health over a wide area.
- Surveillance: They are used in surveillance and in law enforcements.

The main problem with drones is:

- Flight duration: If the battery is not charged properly, the drone might lose control at any time. Also, the battery has to be properly maintained for longer lifespan.
- GPS Signal Lost: At some places, the signal might be lost hence losing control over the drone.
- Wrong calibration: If the compass is not calibrated properly, then the drone might fly unexpectedly. The scope of drones is vast.
- Delivery: With the rise in e-commerce, drones can be used to deliver packages much faster and hence reduce traffic on land.
- Agriculture: Better technologies are required to monitor vast regions of crop fields.
- Filmmaking and photography: Drones are being used and there are more areas that need development such as autonomous event coverage.
- Law enforcement: Drones can be used in law enforcement.
- Search and rescue: Drones with their ability to reach almost any terrain can be effectively used for search and reduce operations.

The main research areas are:

- UAV autonomy: Improvement of flight control. Collision and obstacle avoidance. Better path planning. Communication and improved sensors.
- Mapping: Using the images from drones to map the surrounding.
- Material of the body: Better materials lead to increased flight duration, control and life span.
- Hybrids: Design of hybrid models that incorporate all the pros of fixed wing drones and VTOLs, such as vertical take-off and landing, hover, maneuverability, agility and flight duration.

The areas that I think drones can be improved are:

- Improved flight duration in multi rotor drones.
- Improved payload capacity.
- Ability to manage remaining flight after accidents.
- Improvement on SWARM technologies.
- More agility in multi rotor drones.

Limitations of drones:

- They cannot be deployed during all-weather conditions. Highly dependent on wind speed and rain.
- Speed of the drones is not very great, usually around 30mph.
- Laws restricting it and safety regulations.
- Drones can cause injuries to birds and other animals or property upon collision.
- Drones can easily breach privacy.

Making of remotely controlled drones are easier when compared to autonomous drones. As discussed in earlier Task a drone basically consists of motor, propeller, controller, PDB, ESCs, battery, telemetry (to process real-time data), FPV camera. The above-mentioned components are some basic required components irrespective of autonomous or remote-controlled drones. For a remote-controlled drone, we will use receiver and transmitter in addition to the basic components. Receiver will be connected to the Controller and calibrated with transmitter and the ESC so that there will be proper regulation of voltage for appropriate throttle. There are many important

factors to decide when choosing the controller when it comes to autonomous drones, they are:

The ground-station software

For using ground station softwares like Mission Planner, only AutoPilot hardware controllers can be used. Few of the AutoPilot hardware's are Pixhawk, Beagle Bone (Linux), few of the CUAX controllers, PixRacer etc.

The compatibility for Vision

For implementing computer vision, not all controllers will have inbuilt access for running vision. Thus, we will be needing additional controllers like raspberry. Pixhawk is a one such controller which gives port for raspberry connection. Navio2 which actually has raspberry pi as the core controller and Navio2 acts as a shield.

Picking the right firmware that suits the needs is a daunting task. There are dozens of flight controllers out in the market to choose from, where each is better in their own way. With advancement in technology flight controllers have evolved greatly over the years with faster more powerful and better micro-controllers being used for better optimization of resources and features.

Opensource CC3D Controller

The CC3D boards is an all-in-one stabilization hardware flight controller, which runs the OpenPilot firmware. It can fly any airframe from fixed wing to an octocopter and is configured and monitored using the powerful and user friendly OpenPilot Ground Control Station (GCS) software. The CC3D is compatible with both its native software, OpenPilot, and with CleanFlight. The CC3D is compatible with both its native software, OpenPilot, and with CleanFlight.

Naze32

The Naze32 board has been gaining popularity over the last few months, especially with the 250-size racing crowd. One of the great features of this board is that it's compatible with CleanFlight, an extremely nice software package that is very intuitive and easy to use. The board comes with a micro-USB port that can be used to connect it to a computer. The Naze32 board is available in two versions: the Acro version (around $30) and the Full version (around $50). The difference between the two is that the Acro version has gyros and an accelerometer, while the Full version includes as additional features a barometer, a compass and the provisions to add more advanced functions like GPS. Both versions also support FrSky Telemetry.

Hobbyking KK2.1

This is the "tried, tested, and true" board among flight controllers--it was around for a while and is used by many beginner and intermediate hobbyists. Although it doesn't have some of the advanced features that some newer flight controllers have, it's really nice for beginners due to its ease of programming. (Programming is done by using the four black buttons to adjust the settings that are displayed on the LCD screen). However, there are a couple of downfalls. The board only uses an 8-bit processor, which means that it is slower at issuing commands and thus is not as locked in as some newer boards. This also means that it's not that good for small, fast multirotor like 250-size racers. The best use for this board is for "trainer" multirotor like certain configurations of the Electrohub from Flite Test as well as experimental VTOL airplanes like the Chimera. Another problem is that the board *has to be tuned.* Sure, it's true that most (if not all) flight controllers have to be adjusted in one way or another, but chances are that out of the box the KK2 board's settings will be nowhere near to how you want them. Many pilots state that they are yet to find a frame that works with the KK2's pre-set settings. Also, the KK2 board should definitely be flashed with one of the available firmware's to improve performance.

ArduPilot Mega APM 2.5

Ardupilot Mega is a professional quality IMU autopilot that is based on the Arduino Mega platform. This autopilot can control fixed-wing aircraft, multi-rotor helicopters, as well as traditional helicopters. It is a full autopilot capable for autonomous stabilization, way-point based navigation and two-way telemetry with XBee wireless module. Supports 8 RC channels with 4 serial ports.

The APM 2.5 is a very versatile flight controller that can be used in a variety of applications. It's a great way to equip your aircraft with GPS without breaking the bank. This flight controller comes with various telemetry and logging functions, allowing you to view numerous parameters about your flight after landing. An interesting property of the APM board is that the default settings for the Acro mode still retain some degree of stabilization. Until adjusted, the board will still limit the aircraft's bank angle, return the aircraft to level flight when the transmitter sticks are centered, and keep the bank angle (as opposed to the banking speed) of the aircraft relative to the movement of the sticks. Fixing this issue is a simple change in the board's settings through the APM software.

Another potential problem is the significant drift encountered when in the GPS lock mode. After repeated attempts to re-calibrate the GPS, I managed to obtain some improvement in, yet could not fully eliminate, the drift.

Vector FC

The Vector FC combines all the equipment that you need for autonomous flight into one package. It includes a flight controller with a GPS system for navigation, an OSD (on-screen display) for live telemetry transmission on your FPV monitor, and several sensors that measure various parameters while you're flying. To complete an LRS (long-range system) setup with this unit, you'll need a camera with a live video link, a UHF (ultra-high frequency) receiver and, of course, your aircraft! Note that the Vector can be used on fixed-wing aircraft as well as multirotors. Programming is easily done through a computer interface. This is an excellent choice of flight controller as it gives you all of the benefits of an OSD without the hassle of hunting down every component discretely.

A ground station is typically a software application, running on a ground-based computer, that communicates with your UAV via wireless telemetry. It displays real-time data on the UAVs performance and position and can serve as a virtual cockpit.

Mission Planner

The easiest way to get into autonomous flight is to use Mission Planner as it contains a large array of things you can do with your aircraft. Autonomous flight in Mission Planner falls into two main categories; pre-planned missions (auto mode), and live missions (guided mode). The flight planner screen in mission planner can be used to plan a flight consisting of waypoints to visit and actions to perform such as taking photos. Waypoints can either be chosen manually, or the auto waypoint tool can be used to generate missions to survey an area. Once a mission has been planned and sent to the drone, the Auto flight mode can be used so that the aircraft will autonomously follow the pre-planned mission. Here is a handy guide about planning missions.

Guided mode is a way of interactively commanding the UAV to do certain things. This is done by using the actions tab in Mission Planner or by right clicking on the map. The UAV can be commanded to do many things such as takeoff, return to launch, and fly to a chosen location by right-clicking the map at the desired location and selecting Fly to Here. Failsafe are an important thing to consider during autonomous flight to ensure that if things go wrong, the aircraft is not damaged and people are not injured. Mission Planner has a built in Geo-Fence function which can be used to limit where the UAV can fly and stop it from going too far away or too high. It may be worth considering tethering the UAV to the ground for your first few flights as another backup. Finally, it is important that you have your radio transmitter on and connected to the drone so that, if necessary, you can switch out of the autonomous flight mode into a manual flight mode such as stabilize or alt-hold so that the UAV can be safely piloted to land.

APM Planner 2.0

APM Planner 2.0 is an open-source ground station application for MAVlink based autopilots including APM and PX4/Pixhawk that can be run on Windows, Mac OSX, and Linux. It has a smaller user base and a reduced feature set when compared with Mission Planner.

MAVProxy

MAVProxy is a fully functioning GCS for UAV's, designed as a minimalist, portable and extendable GCS for any autonomous system supporting the MAVLink protocol (such as one using Ardupilot). It is a powerful command-line based "developer" ground

station software. It can be extended via add-on moduled or complemented with another ground station, such as Mission planner, APM Planner 2 etc to provide a graphical user interface.

QGround Control

QGroundControl provides full flight control and mission planning for any MAVLink enabled drone. Its primary goal is ease of use for professional users and developers. All the code is open-source source, so you can contribute and evolve it as you want.

UgCS

The Android application UgCS for DJI is designed to establish a connection between DJI drone and UgCS for the desktop to upload and carry out a previously planned mission, or to fly the drone in "Click&Go" mode. UgCS for DJI can also be used as a stand-alone application to manually fly DJI drone and capture photos or videos.

Multirotor are UAVs with more than 2 motors with blades which help in lift. By adjusting the speed of the motors, ascending, descending and hovering of the UAV can be controlled. By varying the speed of the motors, we will be able to change the direction of the UAV. One advantage of Multirotor UAVs compared to fixed-wing drones are the VTOL operation.

I) Types of Multirotor

There are many types of multirotor including:

- Quadcopter
- Hexacopter
- Octocopter etc.

More the rotors, more stable will be the drone. It will also have a safe landing if in case a wing fails. More current will be withdrawn and thus high-capacity batteries will be a necessity. But more the rotors will also enable more thrust, thus can have more payload too. Quadcopters are highly maneuverable, cheap and faster compared to other types of multirotor. So, depending on the need, the multirotor has to be chosen, for high payload we can go for more wings and for less payload but more speed/ maneuverability we will go for quadcopters/ lesser wings in general.

II) Dynamics of Multirotor

Here the dynamics of multirotor can be understood with the help of a quadcopter. In order to maintain the pose in flight and remain stable, there will be a lot of sensors attached to the multirotor. Few of them are listed below:

- IMU Sensor: It is a combination of accelerometer measuring the linear acceleration and gyroscope measuring the angular velocity in the axes- roll, pitch and yaw. Also, magnetometers are present which measures the earth's magnetic field, thus serves to give reference direction.
- Barometers: Measures the air pressure and hence helps FC to predict height from the ground.
- Ultrasonic sensors: Give precise height from ground. Extremely useful within a range of 50cm from the ground. Beyond that limit, we will have to work with barometer.
- GPS Receiver: It enables locating using GPS and other satellite positioning systems.
- Time of flight sensor: Uses laser to estimate distance.

II.A) Quad Motion

Flight dynamics is the study of the performance, stability and control of vehicles flying through the air or in outer space. It is concerned with how forces acting on the vehicle influence its speed and altitude with respect to time.

Green – Normal Speed, Red – High speed. Moving a quad is done by manipulating angular velocities:

- Thrust- Reducing the velocity of all 4 motors will lower the quad, increasing all 4 motors velocity will increase the height of the multirotor with respect to ground.
- Pitch- Changing the motor speed in front and back, we can move the quad forward and backward. Increasing the forward motors velocity will move the quad backwards and vice versa,
- Roll- To move the quad left and right. By increasing the speed of left motors, the quad will move right and vice versa.

• Yaw- To rotate left and right. By changing the speed of alternate motors, the quad will yaw.

Architecture of firmware (Ardupilot and Pixhawk)

Ardupilot is an opensource autopilot software for drones and other autonomous systems. This will be put in the Pixhawk, which is an Autopilot hardware board enabling active interaction with other components in the drone/ UAV. Companies like NASA and Intel use Ardupilot in there tasting and development phase. It has been consistently under development from 2010.
It is the only autopilot software capable of controlling almost any vehicle system imaginable, from conventional airplanes, multirotor, and helicopters, to boats and even submarines. It is made of the following:

- ArduCopter for copter
- ArduPlane for plane
- ArduRover for rover or grounded vehicles
- ArduSub for AUVs or autonomous underwater vehicles

ArduSub codes where derived from ArduCopter code. Pixhawk is the broadly used Autopilot board due to desirable sensor redundancy and flexible external expansion.

General Architecture / Flow

Ground-Station/ computer -> Communication layer -> FC code-> OS -> Hardware/ FC and Sensors. Ground-Station softwares- Mission Planner is one good option.

Communication Layer- Mavlink is a protocol used by Ardupilot for communication with ground-station softwares/ computers. Mission commands will be stored here in EEPROM and will be executed when put in autonomous mode sequentially.

FC Code- It consists of 3 layers. They are:

- Vehicle Code
- Libraries
- Hardware abstraction layer

Vehicle code

Determined based on the vehicle type that's copter, plane, rover, UAV etc. There are some 50codes for each vehicle type open sourced with majority C++ code and codes carrying the dependencies in the repository. Few of the basic observed codes are the surface tracking, navigation, terrain, takeoff, standby, stabilization, esc calibration, etc.

Libraries

Libraries are code consisting of useful information required for the vehicle types. The libraries include: Sensor drivers, pose estimation, control codes etc. Few of the libraries are explained below:

PID

Proportional-Integral-derivative controllers is a closed loop system. P controller is used primarily to stabilize unstable processes. Main advantage is to reduce the steady state error of system. P with D controller increases the stability of the system by improving control, since D can be used to predict the future with the help of previous error and the gain term(kd). PID is the optimum controller eliminating overshoot, increasing stability and oscillations after tuning the gain terms properly.

Attitude control

It is the process of controlling the orientation of the vehicle type with respect to the inertial frame of reference. To control vehicle attitude sensors like IMU are used, which helps in determining the orientation of the vehicle.

Fence

Fences are the geofences that are location-based service in which an app or other software uses GPS, RFID, Wi-Fi or cellular data is used to trigger a pre-programmed action when a mobile device or RFID tag enters or exits a virtual boundary set up around a geographical location

HAL

HAL are hardware Abstraction Layer, which determines the way Ardupilot is made portable to a lot of different platforms. It consists of a set of headers that defines classes and methods that should be

implemented if Ardupilot should run in a new device/ architecture.

OS- Linux, ChibiOS, BusyBox linux are some of the OS compatible with Ardupilot.

Sensors- The sensors will read the values as digital/ analog values and will transfer the data to the flight code, in particular to the Libraries part, where the respective library package will process with the values produced from the sensor and pass it to the OS again. The OS will ultimately send to the hardware that's the controller which will regulate the flow of voltage to the respective motors accordingly, thus creating a cycle of events.

Gazebo – Robotic simulator usually used along with ROS. Using ROS, we can import our vehicle type into Gazebo and then do various simulations in it like altitude finder, GPS location finding and traversing accordingly, PID tuning giving disturbances, etc. ROS is to be installed in Ubuntu, a popular Linux distribution web server. So, all the libraries and vehicle types can be pre-tested using Ardupilot virtually, resulting better efficiency and analysis done economically. All sensor data can be generated with noise too, and studied and analyzed.

Building Ardupilot in Pixhawk

Ardupilot can be built in windows/ Linux for Pixhawk using various functions like Make, QtCreator. They primarily help in editing the Ardupilot code and building for Pixhawk on Windows or Linux. But both Make and QtCreator are no longer supported by Pixhawk.

Common **sensors** used in drone are:

• IMU - Inertial Measurement Unit. Helps in determining the linear acceleration, angular acceleration of the drone.
• Camera - Uses a technique called optical flow, where frame by frame images will be captured and compared to study and analyze the
• Ultrasonic Sensor - Helps in determining vertical distances. Very accurate. But has a range of maximum 50cm only. So, beyond that we will have to go with other sensors
• Pressure Sensor- Measures the air pressure. Drones use this to determine the altitude with the help of pressure differences

Actuators are motors that converts energy into torque and makes/ controls the movement of the machine/device, in this case the drone.

The steps involved / cycle is explained below:

• A setpoint is fixed. Setpoint also called as SP is the desired position set beforehand.
• The controller plans a way to traverse, which controls the motor via regulating the flow of voltage.
• Then, the current position will be checked, which will be called the process variable or PV. The PV will be calculated with the help of the sensors discussed earlier including IMU, Camera and Pressure Sensors.

Now, the PV and SP will be processed together and error will be calculated and will be adjusted in the controller. There are several controller algorithms including PID which will be explained in detail in the coming pages.

• Thrust- Reducing the velocity of all 4 motors will lower the quad, increasing all 4 motors velocity will increase the height of the multirotor with respect to ground.

• Pitch- Changing the motor speed in front and back, we can move the quad forward and backward. Increasing the forward motors velocity will move the quad backwards and vice versa,

• Roll- To move the quad left and right. By increasing the speed of left motors, the quad will move right and vice versa.

• Yaw- To rotate left and right. By changing the speed of alternate motors, the quad will yaw.

Quadcopters are an under-actuated system as it has only 4 actuators(motors) to have a degree of freedom of 6. The 6 degrees include up/down, right/left, forward/backward, yaw, pitch and roll.

The drag, weight and thrust are the major forces acting on drones.

M1 = Thrust + Yaw + Pitch + Roll
M2 = Thrust - Yaw + Pitch - Roll
M3 = Thrust - Yaw - Pitch + Roll
M4 = Thrust + Yaw - Pitch – Roll

It is called Motor Mixing Algorithms. These are the motor speeds. This thrust, yaw, pitch and roll are the speeds assigned. These speeds can be passed through a PID controller and will tune and enable proper functioning.

The drone takes the four-motor speed as input and the output will be state of the drone/ the position to reach.

An altitude and position control system were designed in this video.
Let's assume we will want to hover the drone at a particular position. That means the thrust exerted by the 4 motors will be equal to the weight of the drone. But, due to external noises such as wind there will be disturbances caused which will constantly move the position of the drone. Thus, a control system has to be adapted to solve this issue. PID controller was chosen in this case.

So, to explain in simple words we will find the error in roll, pitch, thrust and yaw individually and use it with the PID controller to counteract the disturbances. Error is the difference between Process variable (PV) and Set Point (SP).

Error (E) = PV – SP
P = E
I = I + E
D = Prev_Error – E
PWM = kp x P + ki x I + kd x D

Kp, ki, kd are the gain factors which will be tuned using various methods or trial and error method. There is also a plotting tool exclusively called PlotJuggler which helps in plotting and tuning the gains in real time using Gazebo.

This are the basic equations for finding the PWM value. PV means the current position resulting due to disturbances and SP is the desired position. Similar to the equations mentioned above, we will find the PWM value for yaw, pitch and roll and apply it in the Motor Mixing Algorithm mentioned in 2nd page. And once finding the motor speeds we will have to check the boundary conditions, that is t has to be greater than 0 and less than 2x depending on the bytes. So, after these conditions are taken into account, we can send the motor speeds from the controller to the ESC to the motors.

But then comes another problem. When disturbance is made, the drone just seconds before retaining its previous place via roll or so, will have a thrust in an angle which will drift the drone by a little each and every time it alters. Thus, we will be introducing position controller here to solve this issue. Here instead of we assuming the initial position angle references, we will use the position controllers to that for us and send it as the output to the PID loop of roll pitch and yaw mentioned earlier. We will be using the 4 sensors discussed in the first sheet for the position controller.

There are codes to be written for all libraries as seen in Task2 including PID, Fencing, IMU etc. Similarly, we will work on Flight code in this section. There are 2 ways to write this flight code:

• Writing the C code manually, compiling and loading onto the drone. But tuning will be difficult as we will have to tweak in the hardware.
• Drawing block diagrams in Simulink, then auto code to C, then compile and deploy to drones via Bluetooth or Wi-Fi.

Simulink models are easy to understand and tuned easily compared to working on C every time. Tuning in Simulink will be done using various Simulink tools. There are various Simulink support packages which will help in programming our flight control software.

Safety Flags will be used to shut down the drone if it fails to respond to the commands or malfunctions. Thus, prevents damage to properties and the hardware's of the drone itself.

State estimators helps in converting values to useful information's. That is, in case of air pressure sensors, we will get the value of air pressure, but what we want is the altitude. So, we wilk have to determine the altitude using this air pressure. For these sorts of conversion, we will be using State Estimators.
Fault protections are blocks which will prevent from hardware failures and Safety flags. It helps in setting up thresholds, which autonomously controls/restricts the action of the drones in real-time.

In this video we clearly complete the feedback loop putting all the dots together. We have seen the sensors inputting to the flight controller software and it regulating the motor speed to the respective motors via the ESCs.
Now, the motor speeds are inputted into a Model, which calculates few stuffs and returns the sensor value to the flight controller software thus completing the feedback loop. There are many things a model consists of including:

- Actuators -Mechanics
- Environment: Gravity - Aerodynamics
- Airframe - Structure
- Sensors: IMU

To prepare the model there are various steps to be taken into consideration. We shouldn't overfit the model too like giving in details that will complicate and affect the performance. There will be 2 models for working on, one will be the linear model and other non-linear model. Nonlinear models are great for simulation. Linear models will be used to tune the 6 PID controllers explained in video2. First, we will create a nonlinear model and verify the model with some test cases. Once it reaches considerably with respect to the real-life scenario, we can pass it to the linear model. Then we can use a nonlinear model to verify the performance of the control system. Then we can load into the hardware and try.

PID tuning is done here in sets as follows:

- First, we will tune the outer loops also termed as the cascade loops here. For tuning we will take all the gain terms to have a constant 0 assigned by default.

Then, we will slowly give a range of value to let's take the P controller of Yaw. So, we can tune the P terms gain and then add D and I in order to stabilize it properly for Yaw in the cascade loop control. Then, we can move to the Pitch and Roll in order and assign the values using trial and error. Also, we can use the help of graphs to plot and tune, will be comparatively easier. For plotting in ROS, a tuning tool called PlotJuggler can be used due to its ease of tuning and simple interface.

- Then, once all the 4 PID controllers in the cascade loop is tuned, we can proceed with the controllers in the Position Controller following the same rules as mentioned in the above point.

In MATLAB we can first draft the plant, the PID controller block in our case and give values to the gain terms. We can then go to the PID tuning graph and adjust the response time and check if the desired kind of graph is plotted. Once done, check the proposed gain values at the bottom of the tuning tool and put its value in the plant done earlier. Now, check in real time and adjust accordingly.

In ROS, we can run the PlotJuggler tool, run the error of yaw in the plot and observe the oscillations produced. To reduce the oscillations, we can simply slide the PID node run before hand and thus by reducing the oscillations we tune it.

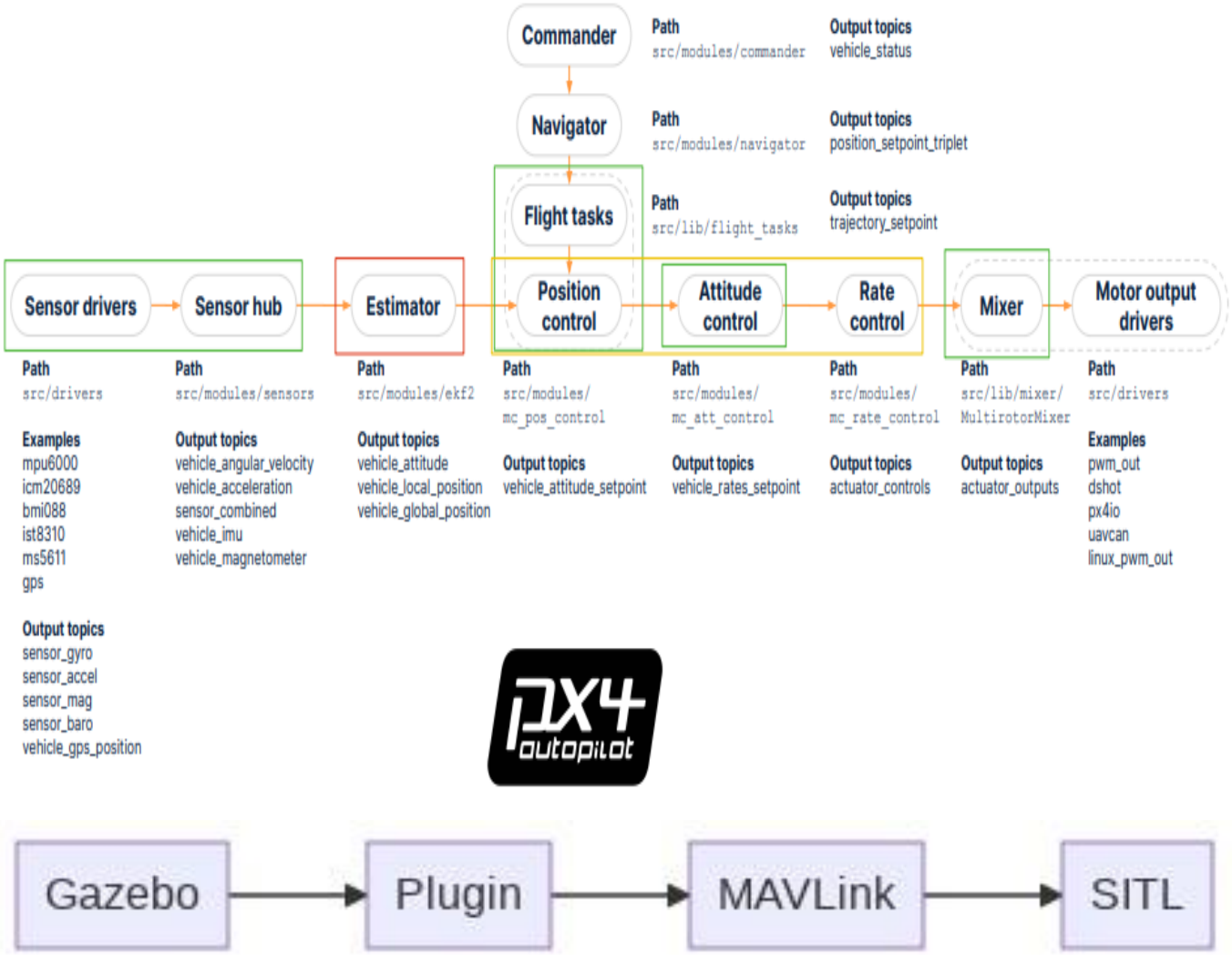
Aerostack (www.aerostack.org) is a software framework for building aerial robotic systems that was developed and used successfully by the Group CVAR (Computer Vision and Aerial Robotics). It is a very reliable, robust, documented and validated software framework.

The main contributions of Aerostack are:
- A completely multi-layered architectural organization to support fully autonomous flights
- A versatile software framework for multiple users; low-level layer for reactive behaviors and high-level layers for intelligent behaviors.

Aerostack Architecture

The Aerostack architecture has 4 important branches:
- Interfaces
- Communication Channel
- Robot Behavior
- Belief memory
- Mission control

Thus, using the above-mentioned behaviors, we can communicate with our aerial we can use this with the help of the behavior tree editor.



Behaviors are simply controlling the robot behaviors thus implementing motion control, planning, swarm interactions etc.

We can first create a Aerostack project by executing few scripts and then selecting the Aerostack modules required for the project. Once the required modules are selected, saved and closed; the modules selected will be downloaded and compiled. After compiling, a txt file will be generated and we can start working on it now.

There are 2 projects using DJI Tello. Steps for one such project is explained below:

Behavior tree for Tello

• Roscore

• Execute the script which launched the Aerostack components for the project

• Open behavior tree editor

• Now, select the behavior based on the tasks. Behaviors include the motion task (take-off, land, hover, wait, etc.), navigation task etc.

ESC AND MOTORS

Many PX4 drones use brushless motors that are driven by the flight controller via an Electronic Speed Controller (ESC) (the ESC converts a signal from the flight controller to an appropriate level of power delivered to the motor).

PDB

A Power Distribution Board is also known as PDB, is an overlooked component simply because it is outdated. PDB's essentially distribute the power from

the battery to the drone esc. But the technology has improved so much in recent days that PDB's also distribute power to some other peripherals such as FPV Video Transmitters, FPV Cameras and the Quadcopter Flight Controller itself. Some modern FC's have integrated PDB's, are limited by space and can only accommodate so much that they do not do a very good job at filtering the voltage spikes from the insane current draws from our quads.

CAMERA

Camera Is highly preferred considering its lightweight, less computation, less heating observed and pixel-based mapping ability.

STEREO

No need for depth correlation. Can directly be obtained from the stereo setup. No external sensors will be required.

MONO

External sensor required for finding the scaling factor. This scaling factor could be found using speedometer. Also, it is computationally quite cheaper as it has to process only one frame per instance and makes the system light weight.

RGBD

Tough calibration for RGBD Cameras. 2 basic calibration required are:

- Calibration between RGB and depth Camera
- Pixel disparity has to be calibrated

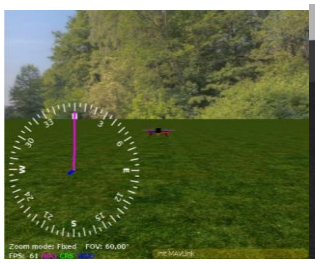
Some famous RGBD Cameras used include the Microsoft Kinect.

PIXHAWK

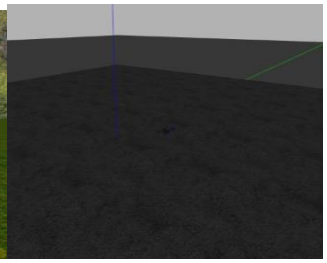
Pixhawk or PX4 is an opensource autopilot flight stack. PX4 is used in a wide range of use-cases, from consumer drones to industrial applications. It is also the leading research platform for drones and has been successfully applied to under water vehicles and boats.

GAZEBO VS JMAVSIM

Gazebo can be used with any robot but jmavsim can be used with aerial vehicles only. Gazebo is easier to use with ROS. Jmavsim is easier to configure and test.



make px4_sitl jmavsim



make px4_sitl gazebo

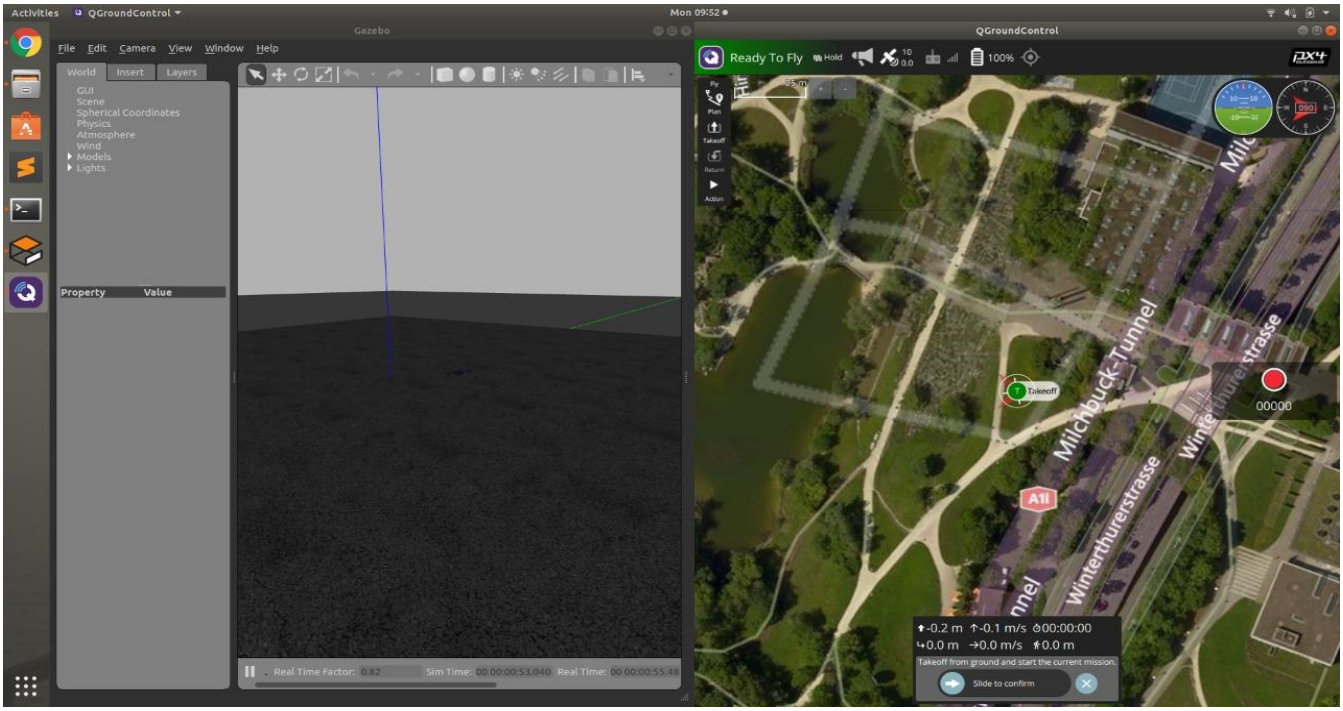
QGroundControl

QGroundControl provides full flight control and mission planning for any MAVLink enabled drone. Its primary goal is ease of use for professional users and developers. All the code is open-source source, so you can contribute and evolve it as you want.

ROS

ROS- The Robot Operating System (ROS) is an open-source framework that helps researchers and developers build and reuse code between robotics applications. ROS is also a global open-source community of engineers, developers and hobbyists who contribute to making robots better, more accessible and available to everyone.

- ROS was built with cross-collaboration in mind. The base code and knowledge can be applied across all robotics platforms (arms, drones, mobile bases, etc.) You'll be able to reuse what you already know and stop reinventing the wheel.
- It can be built into your product. Most of the core ROS 1 packages are under a BSD license and ROS 2 packages are under Apache. This license allows for the modification of code for commercial purposes without having to release your code with an open-source license.
- ROS robots can speak any language. You can communicate easily between Python and C++ nodes, get libraries to allow you to use most



other languages or install rosbridge and use any language that can speak JSON.

- ROS is here to stay. ROS and the community around it have been growing since 2007 thanks to contributions from an incredibly smart and open community. With the market share ROS has acquired and the ongoing development of ROS2, robots will be the future.
- There is a package for everything. Whether you want to compute trajectory, conduct SLAM algorithms or implement remote control, there's a ROS package for that.
- Digital twinning. ROS allows developers to easily simulate their robot in any environment, before deploying anything in the real world. Tools like Gazebo even allow you to create simulations with robots you don't possess.
- It's open-source. ROS has contributors all over the world using ROS for countless different purposes. Every contribution feed into continuously developing, leading stack that ROS is.

The main mechanism used by ROS nodes to communicate is by sending and receiving messages. The messages are organized into specific categories called topics. Nodes may publish messages on a particular topic or subscribe to a topic to receive information.

Master- The ROS nodes are typically independent programs that can run concurrently on several systems. The ROS Master provides naming and registration services to the nodes in the ROS system. It tracks publishers and subscribers to the topics. Communication is established between the nodes by the ROS Master.

Nodes- Nodes are processes that perform some computation or task. The nodes themselves are really software processes but with the capability to register with the ROS Master node and communicate with other nodes in the system. The ROS design idea is that each node is independent and interacts with other nodes using the ROS communication capability. The Master node is described in the ROS Master section to follow.

Topic- Some nodes provide information for other nodes, as a camera feed would do, for example. Such a node is said to publish information that can be received by other nodes. The information in ROS is called a topic. A topic defines the types of messages that will be sent concerning that topic.

Gazebo- Gazebo is a simulation environment that allows for testing of complex systems –robotic or otherwise. It has many uses, including testing the dynamics of a control system before the system has been actualized. You can also use it to test new code or physical systems that might be too dangerous or expensive to test in the real world.

URDF- The original file for performing simulation is the Universal Robot Description Format, or URDF. This xml-like file type is used heavily in ROS for simulation and testing; it is a supported file type for RViz and other ROS tools. The file is a tree structure of child links (like wheels and arms) connected to parent links (like the chassis) by a series of joints. This file is the description of how your system moves internally, and this will determine how it can interact with the environment. The capabilities of the file however are limited in comparison to the new Simulator Description Format, or SDF. SDF files are what Gazebo uses when performing simulation, and in fact before importing one of your URDF models, it will convert it into an SDF. This file format includes more details like friction, damping, and environment properties like heightmaps and lights that are excluded from the URDF file.

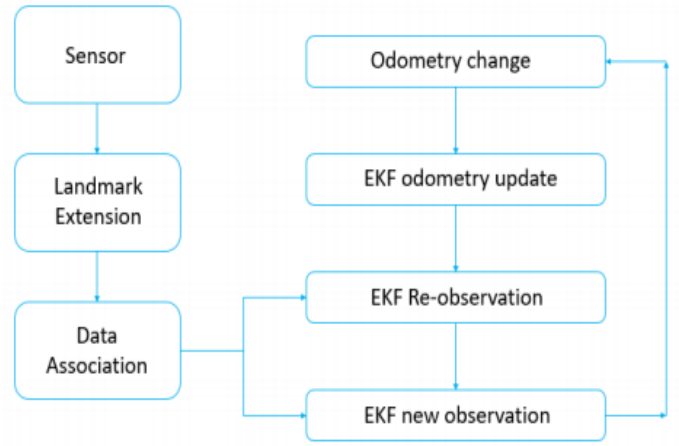
WHAT IS SLAM?

Building maps and localizing a vehicle/ robot in a world at the same time simultaneously is termed as SLAM. It uses Mapping, Localization and Pose Estimation algorithms to build map out of unknown environments. GPS doesn't work well indoors. Even outdoors, it is accurate up to a few meters. This kind of accuracy is too low for autonomous applications. Thus, to address these issues, SLAM was developed. The major advantages of SLAM are:

- SLAM uses many types of sensors like laser and camera (visual also termed as V-SLAM).
- Indoor mapping and location identification are easier with SLAM.
- SLAM can produce 3D images of the surroundings

ARCHITECTURE OF SLAM

SLAM is twofold as the name suggests, it needs to construct or update map of an environment while simultaneously keeping track of the object location.



The basic steps involved in SLAM are:

- Landmark Extraction - The algorithm will terminate only when the population of the particles converging are features which can easily be re-observed and distinguished from the environment. These are used by the robot to find out where it is (to localize itself).
- Data association - The problem of data association is that of matching observed landmarks from different (laser) scans with each other.
- State estimation - The Extended Kalman Filter is used to estimate the state (position) of the robot from odometry data and landmark observations
- State update.
- Landmark update.

Before a robot can answer the question of what the environment looks like given a set of observations, it needs to know from which locations these observations have been made. At the same time, it is hard to estimate the current position of a vehicle without a map. A good map is needed for localization while an accurate pose estimate is needed to build a map.

APPLICATIONS OF SLAM

There are a widespread use of SLAM and some of the common usage to start with are the following:

- Autonomous vehicles/ UAVs
- Autonomous Underwater vehicles
- Planetary rovers
- Domestic robots like Roomba

- Self-driving cars

METHODS OF SLAM

VISUAL SLAM

A technology which uses single camera based on computer vision theory. That's a technology using camera images are called Visual SLAM. In the initial Visual SLAM, the position of the camera was tracked through matching feature point in the image and then 3D map was generated. However, this method has the drawback that the processing speed is slow because matching feature point and updating map has to be performed for all the image frames. As a result, Visual SLAM now uses keyframes to improve performance by parallelizing the tracking and mapping process.

LIDAR SLAM

A LiDAR-based SLAM system uses a laser sensor paired with an IMU to map a room similarly to visual SLAM, but with higher accuracy in one dimension. LiDAR measures the distance to an object (for example, a wall or chair leg) by illuminating the object with multiple transceivers. Each transceiver quickly emits pulsed light, and measures the reflected pulses to determine position and distance. Because of how quickly light travels, very precise laser performance is needed to accurately track the exact distance from the robot to each target. This requirement for precision makes LiDAR both a fast and accurate approach. However, that's only true for what it can see.

LOCALIZATION

EXTENDED KALMAN FILTER

EKF is used to estimate the 3D pose of a robot, based on (partial) pose measurements coming from different sources. It combines measurement from different wheel odometry, IMU sensors and visual odometry. Most real-world problems involve nonlinear functions. In most cases, the system is looking into some direction and taking measurement in another direction. This involves angles and sine, cosine functions which are nonlinear functions which then lead to problems. This issue was the reason to go for EKF from KF. Linear equations give gaussian results when gaussian is applied but nonlinear equations will not give gaussian. And we also know that, in real-world all problems are nonlinear. Therefore, the solution for this problem will be to approximate the nonlinear equations to linear equations. So, after doing this approximation, what we get is the Extended Kalman Filter. This approximation can be done using various tools in statistics like the Taylor series. That's, Gaussian on the nonlinear function will be done followed by taking the mean first and then performing a number of derivatives to approximate it. The first derivative of a Taylor series is called as Jacobian Matrix. This Jacobian Matrix converts the nonlinear curve to linear function. The mathematical steps involved in this EKM are:

- Prediction Step

Estimates of the current position along with noises (the uncertainties).

- Update Step

Nonlinear Measurements coming from the sensor.

$y = z - h(x')$

This is the difference between measured and actual value.

- Mapping Function

Mapping is specified between the Cartesian and Polar coordinates.

$$h(x') = \begin{pmatrix} \rho \\ \phi \\ \dot{\rho} \end{pmatrix} = \begin{pmatrix} \sqrt{p'^2_x + p'^2_y} \\ \arctan(p'_y / p'_x) \\ \frac{p'_x v'_x + p'_y v'_y}{\sqrt{p'^2_x + p'^2_y}} \end{pmatrix}$$

- Kalman Gain

This is basically the weight given to the measurements and the current state estimate.

$S = HjP'HjT + R$

$K = P'HjTS{-}1$

- Jacobian Matrix

$$H_j = \begin{bmatrix} \frac{\partial \rho}{\partial p_x} & \frac{\partial \rho}{\partial p_y} & \frac{\partial \rho}{\partial v_x} & \frac{\partial \rho}{\partial v_y} \\ \frac{\partial \varphi}{\partial p_x} & \frac{\partial \varphi}{\partial p_y} & \frac{\partial \varphi}{\partial v_x} & \frac{\partial \varphi}{\partial v_y} \\ \frac{\partial \dot{\rho}}{\partial p_x} & \frac{\partial \dot{\rho}}{\partial p_y} & \frac{\partial \dot{\rho}}{\partial v_x} & \frac{\partial \dot{\rho}}{\partial v_y} \end{bmatrix}$$

- Polar Coordinates

Thus, using these Jacobian matrices, we will retrieve polar coordinates, that's from cartesian coordinates.

ISSUES WITH EFK:

- If the initial state estimated is wrong, the filter will diverge quickly.
- Impact of imperfect models.

PARTICLE FILTER LOCALIZATION

Also called as Monte Carlo localization (MCL). With a given map, the algorithm attempts to determine the position and the orientation of the robot with the help of particle filters. Particles are basically the location the robot guesses to be present compared to the current position. Thus, this estimated position or the particle is compared with the readings from the sensor data and convergence/ divergence is analyzed. Ultimately, the sensor readings have to converge with the particle (estimated position). So basically, how this works is that the actual and particle positions will be estimated and then probability will be calculated, then when the car starts to move the irrelevant particles will be differentiated and then removed. Note: Before estimating the particle, landmark coordinates have to be fixed based on which rough calculations using the sensor is laid. First let's discuss how to generate the particles. The steps involved are:

- Initializing the landmarks and the world size
- Depending on the world size, random position of the actual robot is estimated.
- Checking if this position is then in the bounds of the world size.
- Find the distance of the robot from the landmark.
- Finally moving the robot and thus returning new position.

Note: Noises are also taken into consideration. Noises indicate the uncertainties caused on the robot on its environment. Secondly, we will discuss how to Measure the Probability. As discussed earlier, probability is determined with the actual position and the particle position obtained. Thus, particles close to the actual robot will have high probability and far-off particles will have less probability value. Once probability is estimated, we will go to the third step that's the Resampling. Resampling is simply removing of the less probability positions or the particles from the hypothesis/ algorithm being processed. The main reason for removing unwanted/ fewer probable positions is:

- Reduces the depletion of the particles.
- Processing with less probable cases/ scenarios is economical
- High probable positions will not be segregated, which increases tracking of useless hypothesis.

Once resampling is done, there will be only one more step to be done, Determining the Orientation. That is, even though we found out the closest location, we will have to determine the orientation of the robot. Let's take a scenario of after 100 iterations, there comes a situation with only 2 particles A and B, A in the direction of the robot itself. So now, when time increases, the probability measured with respect to particle B will be more as it moves away from the robot itself. So, after some more iterations, Particle B will be filtered out. Thus, we can say that when time increases (thereby increasing iterations), we will be able to determine the exact orientation of our robot. Thus, only one particle will be left out which will have high probability which is the particle closest to the location and orientation.

APPLICATION OF PFL

In FASTSLAM, the nomenclature consists of:

- DATA – Input of the robot (go left, right) and the Measurements (distance with respect to the landmarks, etc.)
- STATE – Position of the robot and the landmark

Thus, in FASTSLAM, both the input of the robot and the measurements are calculated at the same time in each and every step. Also, the landmarks are considered stationery. And that the state will have to estimate the value of both the robot and the landmark.

COMPARING EKF WITH PFL

Particle Filter Localization when compared with the Effective Kalman Filter is better based on the following problem; The EKF will represent only one single Gaussian, that's one particular position of the robot. This means that if there are in case two different places where the robot is estimated to be, the EKF will filter one which is most likely to occur. But this goes well only if the chosen position is correct. Otherwise, the complete hypothesis will fail. This is where Particle Filter Localization gives an edge over EKF.

In simple words, PFL will keep in track of all the hypothesis planned and estimates simultaneously and based on the probability estimation done in PFL as discussed previously, it filters out the less likely position (less probability particles) and continues its tracking with new particle observation. This factor as mentioned above makes Particle Filter Localization more effective and robust when compared to Effective Kalman Filter in terms of localization.

HISTOGRAM FILTER LOCALIZATION

Histogram Filter Localization is a grid-based approach that is analogous to midpoint rectangular integration. It is an approximation of Bayes filter. Bayes Filter is an algorithm that estimates the probability distribution of a robot position conditioned on the series of observations that's the camera images and velocity commands. This posterior over states is called as belief. Thus, a Histogram Filter Localization is called as a type or an approximation of Bayes filter as it represents the belief as histogram that's splitting the world with one probability value per state. Also, Histogram Filter Localization is a non-parametric filter. That's it doesn't rely on fixed functional forms. Also, the number of samples biases the speed of the algorithm and quality of the filter in non-parametric filters. The grid will have 2 state variables. One state variable map to the x-axis and other state variable to the y-axis. Once grids are created, the iterations over all the cells of the grid have to be done, do that belief will be updated upon sensor inputs.

if d is a measurement z **then**
  $\eta = 0$
  for all x **do**
    $Bel'(x) = p(z|x)Bel(x)$
    $\eta = \eta + Bel'(x)$
  end for
  for all x **do**
    $Bel'(x) = \eta^{-1}Bel'(x)$
  end for
else if d is a action u **then**
  for all x **do**
    $Bel'(x) = \int p(x|u, x')Bel(x')dx'$
  end for
end if
return $Bel'(x)$

So basically, just like the landmarks in particle filter localization, here in histogram filter localization we will need few references. The robot will move with respect to these references. The robot while moving, will calculate the position probability of the histogram filter, cell after cell in the grid. This filter will integrate the speed input and range observations from the references for the localization task.

DISADVANTAGES OF HFL

The probability of each particular possible state can't be found. We can only be able to find the probability of the state in a certain region of the world.

COMPARING HFL AND PFL

When comparing Histogram Filter Localization with Particle filter Localization, one major difference observed is that with less time, that's a smaller number of particles and a smaller number of grids which depends on the computation speed of the device, the Particle filter Localization is observed to outperform Histogram Filter Localization. Thus, in situations where computational power is severely restricted, the particle filter can outperform histogram filter.

PATH PLANNING

Path planning is one of the basic operations needed to implement robot navigation. So, once we localize our robot as discussed, using various methods like EFK, PFL or HFL, we will have to do path planning. That's once we know the destination point cloud and localize our position, we will have to plan the route the robot/ drone takes in-order to reach the destination. This planning operation is called as Path planning. It can only be done if the map of the world is known beforehand. The path the robot takes has to be collision-free. There are a lot of path planning methods available. In ROS, we use the move_base package which moves the robot from its current position to the goal position with the help of other navigation nodes. The move_base package basically links the local and global planner for path planning. Thus, it will subscribe the goal topic which will be the input of the navigation stack. Once the goal is subscribed, it will be passed to the global_planner, local_planner, recovery_behavior and costmaps which in return will generate the output that's the cmd_vel sending it to the base controller for moving

the robot for achieving the goal pose. Now we will discuss about all the links attached to the move_base. Some of the important links are:

- global_planner –

Path planning is done here using algorithms like A* and Dijkstra also calculates the shortest path possible for traversing.

- local_planner –

Uses odometry and sensor readings and sends appropriate cmd_vel values to the robot controller for accomplishing the plan drafted by the global planner.

- rotate_recovery –

Takes a 360 degree turn and decides the path most suitable to traverse. This is especially used when the robot collides with an obstacle.

- Costmaps –

They create grids on the environment and then navigate accordingly. We know the robot can only plan if it knows the map. Therefore, costmaps create grids on the map and each cell will have a probability which will help the controller to understand if there is an obstacle or it is free.

- Map_server -

It is used to save the map and also load it when needed to navigate.

- AMCL -

Helps in localizing the robot in the map. It basically uses particle filter to localize the robot in an environment with the help of probability theory as discussed earlier in TASK3. AMCL can work only with laser scans.

- Gmapping –

An implementation of the FASTSLAM algorithm. Takes in laser scan data and odometry to build the map of the environment.

Thus, the working of the navigation stack is as follows:

- Localizing the map -Using AMCL / EFK / HFL
- Setting the goal
- Sending the goal and Path planning
- Sending the velocity to the robot controller

NODE BASED OPTIMAL ALGORITHMS

Dijkstra's Algorithm

Finds shortest path in a graph where weights of the edges are already known. It is a form of dynamic programming. It finds the shortest path using local path cost. When applying in 3D space, a 3D weighted graph must be built first; then it searches the whole graph to find the minimum cost path. It has 2 sets, one containing the vertices included in the shortest path tree and other set with the ones not in the tree. The steps involved in this algorithm are:

- Creating a set shortest path tree
- Assigning a distance value to all vertices
- Check the adjacent vertices of the lowest vertex and pick the vertex with minimum distance.
- Update the distance value
- Pick a vertex not in the path tree and check its adjacent vertices
- Repeat the steps until the path tree does include all vertices given.

A-Star Algorithm

In maps the A* algorithm is used to calculate the shortest distance between the source (initial state) and the destination (final state). A* algorithm has 3 parameters:

- g: the cost of moving from the initial cell to the current cell. Basically, it is the sum of all the cells that have been visited since leaving the first cell.
- h: it is the estimated cost of moving from the current cell to the final cell. The actual cost cannot be calculated until the final cell is reached. Hence, h is the estimated cost. We must make sure that there is never an over estimation of the cost.
- f: it is the sum of g and h. So, f = g + h

Thus, this algorithm works in the following way, it will calculate this f value and find the smallest f valued cell and move to that particular cell. This process will be continued until it reaches the destination, that's the goal cell.

D-Star Algorithm

D* resembles A* but is dynamic (its cost can change in the traversing to reach goal). That is, it will assume that there is no obstacle in a particular grid and once it approaches the grid, it will dynamically adjust and adds information to the map and if necessary, re-plans the entire path from the current coordinates. This process is repeated until the robot reaches the destination. There are 3 types of D* algorithm namely original D*, Focused D* and D* Lite. All these are

basically combinations of different path planning algorithms like A*. D* is an incremental search algorithm used for path planning. D* resembles A* but is dynamic (its cost can change in the traversing to reach goal). That is, it will assume that there is no obstacle in a particular grid and once it approaches the grid, it will dynamically adjust and adds information to the map and if necessary, re-plans the entire path from the current coordinates. This process is repeated until the robot reaches the destination. There are 3 variants with the D* algorithm, that's:

- D* - Original Algorithm
- Focused D* - It is a modification of the original D* algorithm. It is a combination of A* and the original D* algorithm.
- D* Lite – An incremental Heuristic search algorithm built on the lifelong planning A* algorithm.

D* algorithm has a lot of benefits:

- Optimal and Complete
- More efficient than A*
- Reduces computational costs
- There won't be much of impact of local changes on the path

Theta-Star Algorithm

Theta * is an any-angle path planning algorithm that is based on the A* search algorithm. They search for a path in the space between two points and takes turn in any angle. The resulting path will be towards the goal with fewer turns. Algorithms like the A* will be limited only within the grids thus will produce indirect and jagged paths. It interleaves smoothening with the search. There are many variants of the theta* algorithm. Some of them are:

- Lazy theta*
- Incremental Phi*

A* VS THETA*

A* algorithms is slower due to many edges and many line-of-sight checks compared to the Theta* algorithm. Slower in terms of constructing the map/ graph. The results of a study concluded that the A* and Basic Theta*algorithm has the same completeness criteria and has time complexity which is relatively same, the A* algorithm has the advantage of optimality in fewer number of nodes searched, whereas the Basic Theta*algorithm has the advantage of the optimality the shortest results.

ROS PACKAGES FOR 3D LOCALIZATION

- Mcl_3dl

A ROS node that performs 3D localization system for robots with 3D LIDARS. It implements point cloud-based Monte Carlo localization aka particle filter localization that uses a reference point cloud as the map. It receives the reference point clouds and localizes the 6 DOF that's the x, y, z, yaw, pitch, roll and yaw poses assisted by the motion prediction from the odometry.

- Hdl_localization

Uses the Unscented Kalman Filter based estimation. It first estimates the sensor pose from IMU data and performs multi-threaded normal distribution transform scan between global point cloud and input point clouds to correct the estimated pose.

- Amcl3d

It is a problematic algorithm to localize a robot in 3D. It uses Monte Carlo localization just like Mcl_3dl. It uses laser sensor and radio-range sensors to localize the UAV with the help of a map. Thus, it with a defined map calculates the weight of the particle and localizes with the probability distributions. It occurs in 3 steps just the way in any particle filter localization that's prediction, update and resampling.

- Rtabmap_ros

Rtabmap stands for Real-time appearance-based mapping. A RGBD SLAM approach based on the global loop closure detection with real-time constraints. It is used to create a 3D point cloud in an environment and also to create a 2D occupancy grid map usually used for navigation. It can be used along with a Kinect, stereo camera or a 3D lidar.

- Dynamic Robot Localization

The dynamic_robot_localization offers 3 DoF and 6 DoF localization using PCL and allows dynamic map update using OctoMap. It's a modular localization pipeline, that can be configured using yaml files.

MOTION PLANNING

GRAPH BASED PLANNING

Grassfire

When a robot moves in a known environment, the floor will be split into grids/ blocks. Coordinates will be assigned to the grids. And then each grid will be assigned a binary value depending on if the grid is empty or not. Now we start from the goal location and mark it as 0 value. Each of the four neighbouring cells that is empty is marked a consecutive value 1. Now the neighbours of the cells that are marked as 1, are given the next value 2. This goes on till the start location is reached. Now each value of the cells surrounding the goal location represents the number of steps required by the robot to reach the goal location from this cell. Therefore, successfully selecting the minimum neighbouring value will lead the robot to the goal location via the shortest path.

Configuration Space

Let as assume a point x in the C-Space. There is a collision detection function called CollisionCheck which will return a binary value depending on what lies in the grid the robot is visualizing. For example, the CollisionCheck will return 1 if there is a collision with an obstacle and 0 if x is a free space. The collision or free space will be found by considering the obstacle and the robot as triangles. Then we will check when the robot is placed at a particular point cloud within the C-Space, if a triangle still prevails or a polygon is formed. If a polygon is formed, it means the robot is coinciding with the obstacle, thus the CollisionCheck will return 1 or if there is only a triangle, it will return 0 thus reflecting it is a free space.

SAMPLING BASED PLANNING METHODS

Rapidly-Exploring Random Tree

The Rapidly-exploring Random Tree is actually quite straight forward. Points are randomly generated and connected to the closest available node. Each time a vertex is created, a check must be made that the vertex lies outside of an obstacle. Furthermore, chaining the vertex to its closest neighbor must also avoid obstacles. The algorithm ends when a node is generated within the goal region, or a limit is hit.

Probabilistic Road Map

A new technique to deal with choosing points. Here, we will randomly choose points in the C-Space instead of uniform selection, assuming we will get the structure of the free space from these random points. Thus, on every iteration, a new set of points are randomly assigned in the C-Space and the CollisionCheck will return 0 or 1 depending on whether free space or collision. Thus, in every turn when it gets a free space, it will try to forge a new configuration and closest existing sample. The problem with probabilistic road map is that there can be a possibility that the robot might fail to find a path even if it exists. Thus, this kind of approach is not considered a complete path planning algorithm.

BIONIC PATH PLANNING ALGORITHMS

Ant Colony Optimization Algorithm

ACO is an intelligence-optimized algorithm that simulates the heuristic mechanism of the shortest route based on pheromone in the process of ants foraging for food. ACO uses all paths of the entire ant colony to describe the solution space of an optimization problem, and obtains the best path through positive feedback based on pheromone. The idea of ACO directly maps the robot's path optimization problem, which is easy to understand and has very intuitive results.

Particle Swarm Optimization

It optimized by continuously iterating to improve the candidate solution with regards to given measure of quality. It will have some particles, which will be moving in the environment/ search space with a particular position and velocity. Each particle will be influenced by its local best-known position which are updated as better positions by other particles. This will move the swarm towards the better solution ultimately. It is metaheuristic as it makes few/ no assumptions about the problem optimized.

Genetic Algorithm

A genetic algorithm is a search heuristic that is inspired by Charles Darwin's theory of natural evolution. This algorithm reflects the process of natural selection where the fittest individuals are selected for reproduction in order to produce offspring of the next generation. The algorithm will terminate only when the population of the particles converge.

SENSOR FUSION

Combining two or more sensor data in a way that generates a better understanding of the system is called as sensor fusion. Thus, sensor fusion does the sensing and perceiving part for the autonomous systems. That is, it will take multiple sensor readings, combining it to get a better model, so that the system can plan and act efficiently.
The major advantages of sensor fusion are:

- Increase in the quality of data by removes constant noises
- It increases reliability (even if one fails, we will get value but not as accurate as before obviously).
- Unmeasurable states can be measured (one camera can't measure the distance between itself and an object, but 2 cameras together can).
- Increasing range (like having many ultrasonic around a car for close distance detection)

Kalman filters are one of the most efficient sensor fusion algorithms used. The advantage of Kalman filters is that the mathematical model of the system is already built in the filter, so fusing of sensors can be done and map quality can be enhanced. Kalman filters will operate in 2 steps:

- Predict
- Measurement
- Update

Predict:

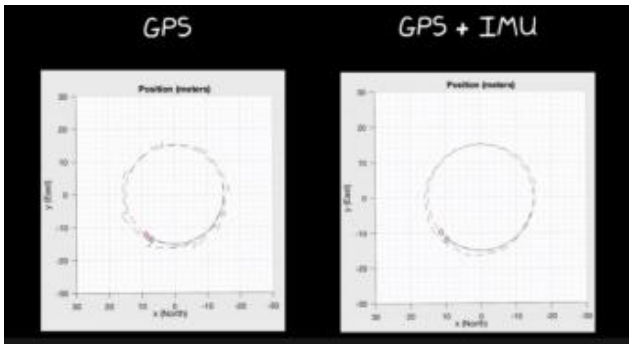
$$p(x_k|y_{1:k-1}) = \int p(x_k|x_{k-1})p(x_{k-1}|y_{1:k-1})$$

Update:

$$p(x_k|y_{1:k}) = \frac{p(y_k|x_k)p(x_k|y_{1:k-1})}{p(y_k|y_{1:k-1})}$$

Prediction is based on the previous knowledge of the robot and kinematic equations. Measurement is getting the readings form the sensors. Update is updating the value based on the predicted and measured readings.

This predict equation uses the position of the previous time-step k-1 together with the motion model to predict the current state xk. This will be updated in the update equation shown above using the Bayes theorem thus combining the measured and predicted states. Now we will have the posterior distribution. Similarly, we can now consider this posterior as the previous posterior and find the next posterior. Thus, this method is iterated for further measurements. It will first measure using a sensor and then uses a mathematical model to compare it with the measured values from the sensors. Thus, it estimates the state from the confidence attained from the predicted and measured values. This way, the sensor will run asynchronized measurements. IMU is a fusion of accelerometer, magnetometer and gyroscope. IMU is often fused with GPS in drones.



In the image seen above, when we use only GPS, the prediction is drastically running away from the ground truth due to the slow update rate of the GPS. Whereas when IMU is added the prediction will be corrected with the fast update rate of the IMU sensor against that of the GPS. Thus, once the sensor readings converge, we will get a better prediction and thus a better state estimation.

LIDAR CAMERA FUSION

LIDAR has an advantage of sharp accuracy in ranging and thus results in good mapping. Coming to Visual sensors, they provide a very dense map and comparatively cheap and consumes less power. But it also has its cons, including poor depth estimation, small range for stereo-visions, difficult to use RGBD cameras outside due to the lightings. Thus, considering both its pros and cons, a fusion of these two sensors would prove to be very efficient in SLAM. Two important considerations when it comes to fusion of sensors are the calibration of the sensors to be fused and the approaches/ architecture of fusing the sensors. One of the approaches of the fusion is:

- Camera resolution is much higher than a lidars but less depth information. Thus, a gaussian process regression was performed to interpolate the missing depth values using the lidar. Thus, Lidars where used to initiate the features declared in the images.
- When the visual tracking is unsuccessful, the Lidar pose can be used to localize the point clouds data of the RGBD camera to build the 3D map.

A few sensor fusion methods are:

- Kalman filtering
- Optimization based methods

Kalman filtering is again divided into 2 types:

- Loosely coupled (fuse processed information's from individual sensors)
- Tightly coupled (fuse raw measurements directly)

Optimization-based methods are mostly tight-coupled. The advantages of using Kalman filter for sensor fusion are:

- Sensor data fusion can be easily implemented using Kalman filters as it already has mathematical models inbuilt.
- Increases the on-line estimation by reducing the noise and bias.

Camera-GPS-IMU sensor fusion for autonomous flying

The above-mentioned research paper, explains state estimation for aerial robots in particular using the sensor fusion of Camera-GPS-IMU using EKF filters for ease and robustness. Generally, the state estimation of an UAV is done with the fusion of GPS and IMU. But GPS has a lot of disadvantages. Some of the major disadvantages are slow update rate, low accuracy with position drift. Thus, the paper suggests using camera thus fused with the IMU-GPS combination to increase the update rate and position drift of the estimation process. One notable advantage with this fusion of camera along with GPS and IMU using EKF as observed from the experimental study of the author is that, even when GPS fails due to some reception glitches it was observed that the IMU-Camera fusion gave very reasonable estimates that is very close to the IMU-GPS-Camera fusion. Thus, the author concludes by saying that the Camera-IMU-GPS sensor fusion can enhance accuracy and robustness of the aerial robots.

"Vision-Controlled Micro Flying Robots from System Design to Autonomous Navigation and Mapping in GPS-Denied Environments"

In this paper the author first implemented Visual EKF based SLAM. But the absolute scale in this technique has to be manually set by the user. Thus, the system was extended by adding an IMU sensor. Thus, the estimate of the absolute scale will be automatically set while calibrating. Also, the author used GPS only in the starting of the system to initialize the scaling factor with respect to the ground truth. The main usage of GPS in the beginning was to align all the states for a simpler comparison with the ground truth and also that it would be done quickly. GPS was no longer used as input in the EKF framework apart from the initialization phase.

VISUAL SLAM

Visual SLAM is a type of SLAM that leverages 3D vision to perform localization and mapping of the unknown environment. Thus, using visual inputs of the camera, the position and the orientation of the robot with respect to its surrounding is calculated and mapped simultaneously. Visual Odometry (VO) is the process of estimating the motion of the robot with the help of sequence of images of the environment. VO is divided into monocular and stereo camera methods which can be further divided into feature making, feature tracking and optical flow techniques. Feature tracking is the process of tracking and monitoring particular features in between adjacent images like the window or a door in a particular image. It is basically used to monitor variations in motion in between frames. Feature matching is the process of extracting individual features and matching those features over multiple frames. They are significantly useful when there is a change in the appearance of the features occurring due to monitoring over a longer sequence. Some of the feature extraction techniques are:

- Harris Corner Detection
- FAST Corners
- SIFT Features
- SURF Features
- KAZE

- AKAZE
- BRISK

FAST CORNER

It is one of the fastest feature extraction technique which could be used for extracting features and then to track and map. It is best for live real-time application point of view with efficient computation.

METHOD:

It takes a pixel (p) from the image and circle it with 16 pixels called as the Bresenham circle as the first step to detect corners. Now we will label each pixel from 1 to 16. Then check random N labels in the circle if the intensity of the labeled pixel is brighter than the pixel around which 16 pixels has been selected. Thus, the conditions for the pixel p to be a corner is that:

- The intensity of x should be greater than the intensity of p + threshold
- Or the intensity of x should be less than the intensity of p – threshold

The main significant step in this method is the point where we decide the N and the threshold. N is the pixels out of the 16 pixels to be considered. More the N, more accurate but more computation. N is usually taken as 12. So, this trade-off has to be considered smartly to get desirable results. The difference between the pixel p and the surrounding N pixel values is calculated and stored in a variable V. Now 2 adjacent keypoints will be considered and its corresponding V value will be calculated. The lowest V value will be castoff. This way multiple interest points will be avoided.

DISADVANTAGE:

It will not work properly if there are no contrasting pixels around the center of the pixel p. Thus, we will have to add a blur using Gaussian filters. This will give less precise but a corner.

The major disadvantages of corner detection algorithms are the scaling invariance. That's when a corner is zoomed in, it will represent an edge more than a corner. Thus, came the SIFT or Scale Invariant Feature Transform which solves this scaling issue. The major advantage of this algorithm is that it is invariant to

- Image brightness, contrast
- Rotation
- Scaling

SIFT

The working of SIFT algorithm can be split into 5 topics:

- Scale-Space Extrema Detection
- Keypoint Localization
- Orientation Assignment
- Keypoint Descriptor
- Keypoint Matching

Scale-Space Extrema Detection

The process involved in this step is described below:

- The image is blurred using Gaussian filters.
- Octave is taken. That's a pixel will be selected and compared with the 8 pixels around it. These 9 pixels are considered to be an octave.
- Now 2 more octaves above and below the pixel we took is taken.
- Now, we will check if there is a local extremum and if yes, then it will be considered a potential keypoint.

Keypoint Localization

This process majorly does 2 important things. They are:

- Removing edges using eigen values and ratios
- Removing the extrema of intensities lesser than the threshold values set by us

Thus, removal of low contrast keypoints and edge keypoints happens in this step.

Orientation Assignment

In this step orientation will be assigned to the keypoints. With these orientations for the keypoints being created, an orientation histogram will be derived from the orientations. And when the orientation is more than 80% for a keypoint that keypoint will be taken into consideration for calculating the orientation.

Keypoint Descriptor

It is the step where vectors are derived from the orientation assignment step. The size of the vector is the number of keypoints * 128. The algorithm will create a 16*16 neighbor around the keypoint in which each subblock will have 4*4 pixels. Now, each of this pixel will have 8 pixels around it. Thus, 8*4*4 gives 128.

Keypoint Matching

It is the process where keypoints are matched between images irrespective of the rotation, contrast or the scaling. This is done by identifying the nearest neighbors and doing ratio analysis between closest and second closest neighbors. Also, if this ratio is greater than 0.8, then the keypoints are rejected. This is done to avoid closest match of neighbors which happens due to noises.

BRIEF

It uses binary strings as feature point descriptor. In other words, instead of the 128-bin vector in the SIFT, we will use binary strings in BRIEF. BRIEF stands for Binary Robust Independent Elementary Features. It easily outperforms SIFT in terms of speed and recognition rate. Here, the neighbors around the keypoint are called patches. Thus, the major function of BRIEF is to identify the patches around the keypoint and convert it into a binary vector, so that they can represent an object. Thus, each keypoint will be described with the help of the binary 1's and 0's. One major consideration about the BRIEF is that it is very noise sensitive as it deals closely with pixel-level images. Thus, smoothening is very important to reduce the noise from the image. Smoothening is done using Gaussian Kernel in BRIEF. Once smoothening is done, the next step is the feature descriptor.

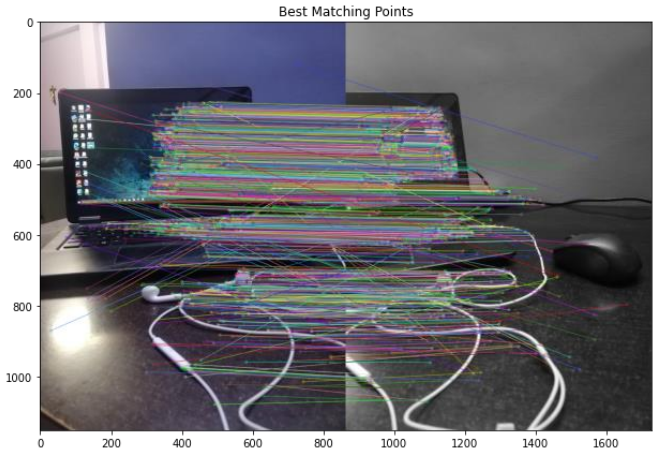
$$Where\ \tau(p;x,y)\ is\ defined\ as:$$
$$\tau(p;x,y) = \begin{cases} 1 & : p(x) < p(y) \\ 0 & : p(x) \geq p(y) \end{cases}$$
$$p(x)\ is\ the\ \text{int}ensity\ value\ at\ pixel\ x.$$

Now the problem will come when we will have to calculate the n (x, y), that's the x, y pairs. This (x, y) pairs are called random pair inside the patch. N is the length of the binary vector and τ is the binary test response of the vector. Finding the random pair can be easily done if the length of the binary vector is decided as we can pick from the patch easily then. There are 5 different methods used to calculate the length. They are uniform (GI), gaussian (GII), gaussian (GIII), coarse polar grid (GIV), coarse polar grid (GV).

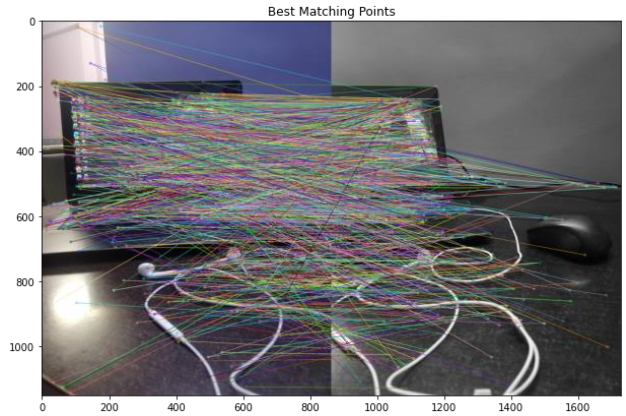
ADVANTAGES- Faster than SIFT in terms of speed and recognition rate. Also, BRIEF relies on less intensity difference between the image patches.

ORB (Oriented FAST and Rotated BRIEF)

FAST algorithm is used to calculate and find the keypoints from the image. Thus, it was most often used to detect and differentiate edges, corner and flat surfaces. It doesn't calculate the orientation just like SIFT method. Thus, thus algorithm ORB, solves this issue. Also FAST isn't invariant of the scaling of the image. ORB also solves this issue. The computation of ORB is done using BRIEF algorithm., Thus it is a mixture of both FAST and BRIEF making it better than SIFT in terms of speed and computation.



Number of Matching Keypoints Between the Training and Query Images: 2455 Implementation with FAST detectors and BRIEF descriptors



Number of Matching Keypoints Between the Training and Query Images: 1871 Implementation using SIFT feature extractors

Development of a real-life EKF based SLAM system for mobile robots employing vision sensing

In this paper, the author used stereo camera (two webcams) for a ground robot to implement the 'observe' part of the system. The 'observe' step consists of image feature selection, tracking of the features selected. The feature selection is selecting of the patches/ windows which can be used for tracking efficiently in subsequent frames. Here, for feature extraction they used a technique called KLT/ Kanade-Lucas-Tomasi Tracker. The major significance of this KLT tracker is that one of the stereo cameras (let's take right) will be used to obtain features via patches/ windows and the camera at left will be used to track the features obtained from processing the image from the right camera. More pixels are chosen as a single pixel will be hard to keep track, due to varies issues like contrast in intensities or confusions due to noises in the data. Thus, N patches/ windows are preferred for tracking the features. This way the features can be selected and tracked.

The next important step is calculating the 3D distance between our location and the feature. The line of sight of both the cameras intersects at a point say for instance P. Now, with this intersection points the point clouds can be obtained.
Once the landmark is identified and tracked, they can be initialized in map utilizing the usual procedure in EKF-based SLAM algorithm. That's the algorithm goes as such:

- Waypoints will be initialized for navigation.
- Traversing of the robot and performing the predict step of EKF
- Observing- Extracting the features and tracking of the features and also 3D distance calculating of the features from the robot.
- Add the features attained to the unknown map of the environment. Then continue the normal observe and update step of the EKF algorithm.
- Now, repeat the same steps again and again updating the new landmarks/ features to the map we started constructing in the fourth step.
- These steps should now be iterated till the final stated waypoint is reached.

VSLAM approaches:

- Feature based approach
- Direct approach
- RGBD Approaches

FEATURE BASED APPROACH

This is again divided into 2 approaches- Filter based and bundle based. Example for filter based is MonoSLAM and for bundle adjustment it is PTAM.

MonoSLAM

In MonoSLAM, camera motion and 3D structure of an unknown environment are simultaneously estimated using an extended Kalman filter. camera motion and 3D positions of feature points are represented as a state vector in EKF and the tracking of this features is observed in MonoSLAM. Various methods for this process of feature detection and tracking have been discussed earlier. The issue with this kind of approach is that when the size of the room/ world increases, the state vector will also increase causing more computation. Thus, BA algorithms were introduced to overcome these issues.

PTAM

It is an BA algorithm/ technique. Bundle Adjustment can be defined as simultaneous refining of the 3D coordinates describing the scene geometry, the parameters of the relative motion, and the optical characteristics of the camera employed to acquire the images. The tracking and mapping are done parallelly to save the computation cost. PTAM is a hybrid of:

- BA part
- Localization with known landmarks
- VO

The basic approach of the PTAM is:

- Initializing the map using the five-point algorithm. Five-point algorithm is essentially taking 5 points in between 2 images and the rotational and translational matrix along with the essential matrix is found.
- Camara poses are estimated from the previously obtained feature points by projecting on the image thus making a correspondence.
- 3D position of the feature points is estimated using the triangulation method.

- Tracking by randomized tree-based classifiers for searching the nearest keypoint.

BA-BASED APPROACH VS EKF-BASED APPROACH

It is known that more the keypoints, more will be the accuracy and robustness. But more the keypoints/ features selected more will be the computation cost. But due to the parallel tracking of the BA based approaches, they can handle a greater number of keypoints than the filter-based approaches with similar computation cost. Thus, BA-Based approaches are more robust than filter-based approaches.

RANSAC

RANSAC stands for Random Sample Consensus. Deals with removing outliers from inliers contained in a data. No real-world sensor readings are perfect. Thus, we use RANSAC which is a simple trial and error method with groups the inliers and outliers separately. Thus, it helps us to throw away the outliers and work with inliers alone which will save our computation and time. It involves a 4-step processing, and they are:

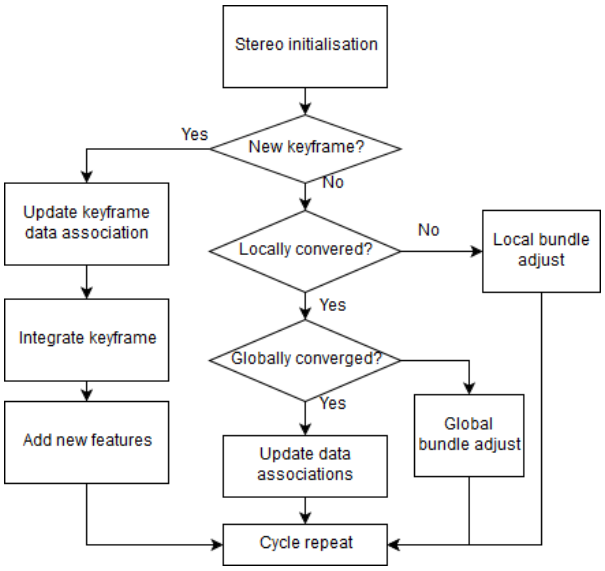
- Sampling
- Compute
- Score
- Repeat

Let's assume there are 2D points (Figure 1) in which a line has to be fitted. Now we will sample and randomly take one line marked in figure 2 as our inlier. Now we will compute the number of supporting inliers satisfying the line we drew. This will be done by parallelly extending imaginary lines on both the side of the line assumed to be the inlier with a uniform distance of **δ.** Now each and every point lying inside these imaginary lines constitute to the scoring of the inliers. Now for the figure 2 there are 4 inliers.

Now we will repeat the steps mentioned above repeatedly and the score will be overwritten with the best score after every iteration. In the 3rd image, that's after some iterations, we got a line which has 12 points satisfying that line model which is the best score to be obtained. Thus, it will be best fit for the line. This way we can eliminate all the outliers easily.

PTAM

PTAM is an BA algorithm/ technique. Bundle Adjustment can be defined as simultaneous refining of the 3D coordinates describing the scene geometry, the parameters of the relative motion, and the optical characteristics of the camera employed to acquire the images. The tracking and mapping are done parallelly to save the computation cost.



5 POINTS OF PTAM:

- Tracking and mapping is separated and run in 2 parallel threads
- Mapping is based on keyframes, which are processed using BA
- The map is densely initialized from a 5-Point Algorithm
- New points are initialized with an epipolar search.
- Large numbers of points are mapped.

TRACKING AND MAPPING

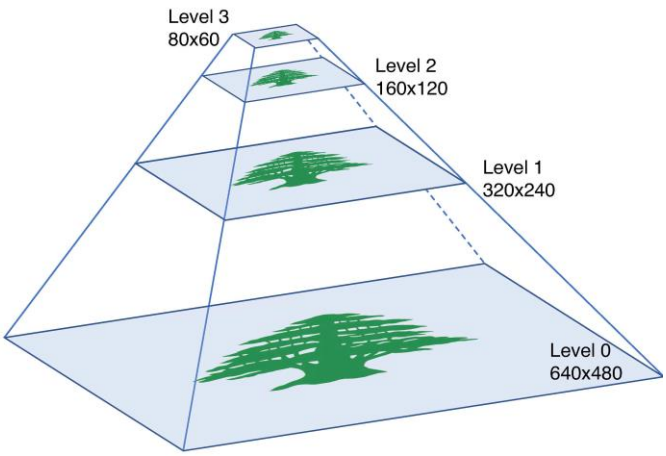
It is done parallelly in different threads (nowadays, all computers have more than one core). The major advantage of processing separately is:

- Tracking will not be slaved to the map making procedure
- More thorough processing can be done as the computational burden to update a map in every frame.
- Every single frame isn't necessary to be used,

as there will be a lot of redundance, especially when the robot is hovering.

Now, we will start explaining tracking and mapping individually with the architecture in detail.

TRACKING



4-level pyramid representation of an image

PTAM generated a 4-level pyramid representation of every frame it obtains as shown in the above figure. It uses this structured data to enhance the features robustness towards scale changes and to increase the convergence ratio of the pose estimates. Each level is formed by blurring the level before it and sub-sampling by a factor of 2.

IMAGE ACQUISITION

The captured images are converted into 8 bits per pixel greyscale for tracking purposes. On each of the pyramid levels, FAST-10 corner detector algorithm is run resulting in a blob-like cluster of corner regions.

CAMERA POSE AND PROJECTION

Projecting map points into the image is vital for tracking. This is done by first converting the world coordinate frame to a camera-centered coordinate frame. This conversion is done by left multiplication with the camera pose.

CW represents frame C from frame W. Jth point of the map is represented by pj. The transformation between the camera-centered coordinate frame and the world is called E_{CW}. Thus, in the first equation we will find a particular point on the map in the coordinate frame W.

$$p_{jC} = E_{CW} p_{jW}$$

CamProj() is a camera calibrated projection model used to projecting the points in the camera frame to the image.

$$\begin{pmatrix} u_i \\ v_i \end{pmatrix} = \mathrm{CamProj}(E_{CW} p_{iW})$$

The below equations show the pin-hole camera projection parameters including focal length, principal points and the distortion

$$\mathrm{CamProj}\begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} u_0 \\ v_0 \end{pmatrix} + \begin{bmatrix} f_u & 0 \\ 0 & f_v \end{bmatrix} \frac{r'}{r} \begin{pmatrix} \frac{x}{z} \\ \frac{y}{z} \end{pmatrix}$$

$$r = \sqrt{\frac{x^2 + y^2}{z^2}}$$

$$r' = \frac{1}{\omega} \arctan\left(2r \tan\frac{\omega}{2}\right)$$

The major requirement of tracking is to observe change in the change in the position of the map with respect to change in the camera pose. This is shown in the below equation which is done using the left multiplication with respect to the camera motion which is represented by the letter M.

$$E'_{CW} = M E_{CW} = \exp(\boldsymbol{\mu}) E_{CW}$$

μ is the 6-vector parameter used here, representing 3 translational and 3 rotation axis and magnitude.

PATCHING

To find a single point p in the map, a fixed range image search is done around the points of the predicted image. To perform this search, the corresponding patch must first be warped to take account of viewpoint changes between the patch's first observation and the current camera position. So, the warping matrix is shown below where u and v are the horizontal and vertical pixel displacement in the patch's source pyramidal level and u_c and v_c are pixel displacement of the current camera frame.

$$A = \begin{bmatrix} \frac{\partial u_c}{\partial u_s} & \frac{\partial u_c}{\partial v_s} \\ \frac{\partial v_c}{\partial u_s} & \frac{\partial v_c}{\partial v_s} \end{bmatrix}$$

The differential of the A matrix tells us in which pyramid level, the patch should be searched.

POSE UPDATE

From the patch observations obtained the pose of the camera can be computed. From each observation, patches pose is found with an assumption of noise measured. Thus, we will calculate the pose by iterating continuously till convergence is observed from any set of measurements.

Obj is the objective function and **σ** represents the robust estimate of the standard deviation of the distribution derived.

$$\boldsymbol{\mu}' = \underset{\boldsymbol{\mu}}{\operatorname{argmin}} \sum_{j \in S} \operatorname{Obj}\left(\frac{|e_j|}{\sigma_j}, \sigma_T\right)$$

$$\boldsymbol{e}_j = \begin{pmatrix} \hat{u}_j \\ \hat{v}_j \end{pmatrix} - \operatorname{CamProj}\left(\exp\left(\boldsymbol{\mu}\right) E_{CW} \boldsymbol{p}_j\right)$$

The pose update is thus computed iteratively by minimizing the objective function of the reprojection error.

TRACKING

At first, a frame will be taken from the live video and a prior estimate for the camera is generated from the motion model. Depending on the prior pose estimate the map points will be visualized. To increase the tracking resilience to rapid camera movement the patch search and pose update will be done twice in PTAM. Both the cycle is explained below:

- A small amt (50) of coarse scale features will be searched in the image. It is done in the highest level of the pyramid representation of the image. From these coarse matched features, the camera pose will be measured and updated.
- A large number of points (1000) are now re-projected from the remaining potential points obtained and searched in the image over a smaller range.

MAPPING

Map building occurs in 2 steps. They are:

- Building the initial map using stereo technique.
- Map will be continuously refined and expanded as new keyframes are added by the tracking system.

These 2 steps are individually explained in the coming pages.

INITIAL MAP

We use 5-point algorithm to initialize the map initially. The initial map will consist of only 2 keyframes taken by user upon which the 5-point algorithm and the RANSAC can estimate and triangulate the base map.

REFINING AND EXPANDING MAP

As the camera moves away from the 2 keyframes formed in the initial map, new keyframes and map features are thus added to the system to allow the map to expand.

REFINEMENTS

BUNDLE ADJUSTMENTS

It is defined as simultaneous refinement of the 3D coordinate describing the scene geometry and the optical characteristics of the camera giving input images. It is basically applied as the very last step of feature-based 3D reconstruction. It is used to remove the noise pertaining the image features observed.

DATA ASSOCIATION REFINEMENT

Once BA has converged, data association will be done. New measurements will be made in the old keyframes. When new features are added, they will be checked if it is an inlier or outlier. If outlier it will be cast off the map. This will be very useful in cases where tracking goes wrong which happens. If the measurements are given low weights by the M-estimator in the BA refinement process, they will be considered to be outlier. A second chance is given by the data association refinement, where its keypoint will be re-measured in the keyframe with a very tight search region. If outlier even after this measurement, it will be cast off permanently. If it is an inlier, it will be added to the map.

This process is given the least priority in the hierarchy. Only if there is no process at hand to be done, this refinement will be done. That's, only if there are no other keyframes to be measured, it is done and once a new keyframe comes in it will be abruptly stopped.

VISUAL INERTIAL SLAM

Adding inertial sensors to the camera to estimate robust pose of a robot is VI-SLAM. IMUs are thus fused to IMU data for localization. It is divided into 2 types:

FILTERING BASED

It is again divided into loosely coupled and tightly coupled depending on the sensor fusion type. The loosely coupled method usually fuses the IMU to estimate the orientation and change in position. The tightly coupled method fuses the state of the camera and IMU together into the motion and observation equation and then performs state estimation.

VI-SLAM has 3 strategies for feature tracking:

- Feature Extraction + Descriptor Matching
- Feature Extraction + Filter Based Tracking
- Feature Extraction + Optical Flow Tracking

Feature extraction in VI-SLAM can be done using Harris, FAST, ORB, SIFT and SURF. When it comes to feature tracking as mentioned above one of the 3 types will be used. Descriptor matching and filter based matching methods model the target area in the current frame and predict positions by finding similar area in the model in the next frame. Optical flow is an efficient way of estimating the motion. It relates the movement in the image brightness mode and expresses the change in an image.

OPTIMIZATION BASED

Some of the optimization-based techniques are:

OKVIS is a keyframe based VI-SLAM technique that will combine the IMU and reprojection error terms into the cost function to optimize the system. For initializing and matching, it acquires the IMU measurement thus having a preliminary uncertain estimate. Harris corner detector combined with BRISK descriptor is used for extracting features. Initially, keypoints are stereo-triangulated and inserted into a local map. Then brute-force matching of the map landmarks are done and the outliers are removed using chi-square test. RANSAC is not used as it is computationally expensive.

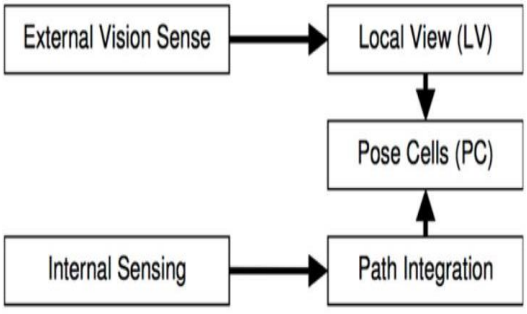
VIORB is a mono tightly coupled VI-SLAM based on ORB-SLAM. Basically, it will contain the front end of ORB SLAM, along with optimization backend, loop closure and relocation. Also, bundle adjustment is used to optimize the keyframes in the local map.

VINS-mono uses the front end of KLT optical flow to track Harris corner, while the backend uses sliding window for nonlinear optimization. Optical Flow recovers image motion at each pixel from spatial-temporal image brightness variation. So, the system consists of estimation, local BA without relocalization, loop closure and global pose optimization. In the front-end RANSAC is used to remove the outliers.

Stereo Multi-State Constraint Kalman filter system (S-MSCKF), which is a state-of-the-art VIO system with low computational cost and good robustness. Synchronized stereo images and IMU messages are fused and a real time 6DOF pose estimation of the IMU frame is generated.

RATSLAM

It is a biologically inspired approach hoping for robotic navigation without the requirement of costly sensors and computationally intensive algorithms. It is a SLAM system based on the combination of appearance based visual scene matching, competitive attractor networks and semi-metric topological map representation. Attractor networks are neural networks that Is designed to converge a stable pattern of activation across its unit. It takes in mono images and odometry as standard messages. It integrates the odometry data along with the landmark sensing using a competitive attractor network thus getting a consistent representation of the environment. The robot thus navigates just like the rat moves in an unknown environment and when it finds a known landmark, the neuron will spike and note the errors and keep in check just mimicking the rat's approach.



Pose Cell

Pose cell encodes the robot's orientation and location at the same time. So, when the robot sees the landmark, it will precisely encode its place using the pose cell and keep errors in check.

Local View

Local View encodes the visual information's in a one-dimensional array of cells. Here, using camera data, locations are estimated as patterns by an appearance-based view recognition. Thus, camera information's are matched with the growing dataset. Once an image feature is recognized, appropriate cell is activated.

Mapping and Re-Localization

Here, the mapping is done by associate learning and the re-localization is done by adding the activation to the pose cell. So once, an activity is found in the local view, the RATSLAM algorithm will send it to the pose cell, by which the correct pose estimate is made.

RTABMAP

RTAB-Map stands for Real-Time Appearance-Based Mapping. It is a Graph based SLAM approach based on appearance-based loop closure detection. It checks how likely an image comes from the previous location. Graph SLAMs have better accuracy than FAST SLAM. Here, we used RTAB-Map with Kinect only for 6DoF mapping. Kinect is a stereo vision sensor with depth camera to determine the depth instantaneously thereby saving estimating computation. Appearance based SLAM means that the algorithm will use the data obtained from vision sensors to localize the position of the robot and simultaneously map the robot in the environment. Loop Closures are used to determine if the robot has already seen the particular frame before. Local Loop closure is dependent of Visual Odometry whereas Global Loop Closure is independent of the estimated pose that's visual odometry. Thus, when the robot moves, the map expands and the number of images that is compared increases in turn. It thus creates dense maps unlike ORB SLAM.

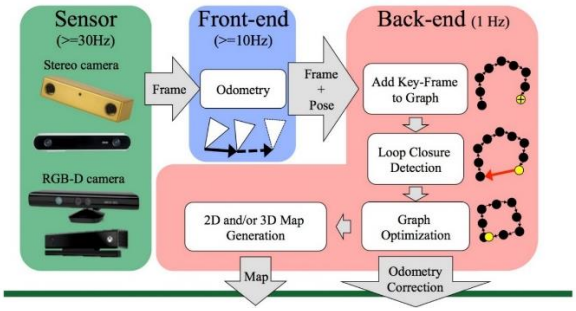


Figure 1. Architecture of RTAB-Map

Working of RTAB-Map

RTAB-Map uses depth image with RGB images to construct maps. The graph is created here, where each node contains RBG and depth images with corresponding odometry pose. The links are transformation between each node. When the graph is updated, RTAB-Map compares the new image with all previous ones in the graph to find a loop closure. When a loop closure is found, graph optimization is done to correct the poses in the graph. For each node in the graph, we generate a point cloud from the RGB and depth images. This point cloud is transformed using the pose in the node. Map is then created.

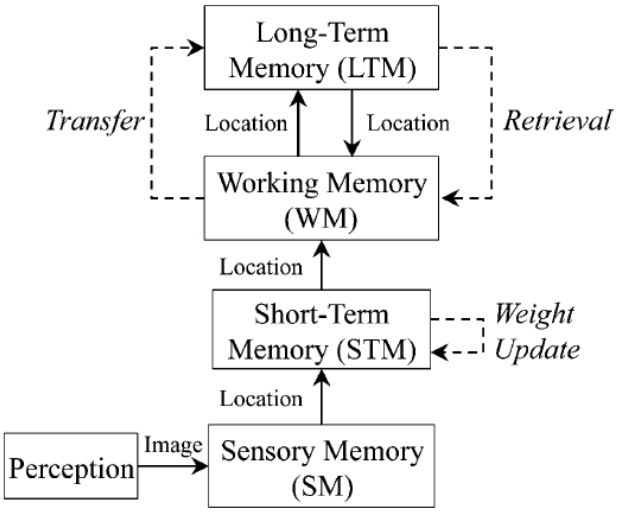


Figure 2. Memory Management

Fig. 2 shows the memory management technique adapted by RTAB-Map. We know that when our robot has to map large environments, the computation increases as the loop closure stores and checks through a hefty database. Thus, for real-time applications, RTAB-Map will become too expensive and less robust. Thus, a memory management method has been adopted to make RTAB-Map robust and usable in large environments in real-time as well. So,

the flow is as follows, perception module to sensory memory from where no subsequent frames are passed to the working memory considering high probability to get similar frames. Based on the location similarity weights are updated for the respective frames. If the loop closure in the takes more time a transfer method is adapted wherein old weighted locations are sent to the long-term memory. Thus, in LTM map is thus stored. Then, new locations are again checked similar to the process as mentioned above. If a old frame matches with the new location then retrieval of the old frame takes place. It will then be fused with the graph map made previously thereby creating the global map.

Implementation of RTAB-Map

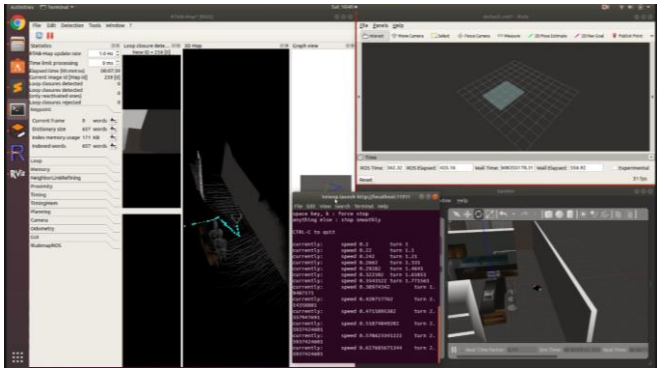


Figure 3. RTAB-Map Implementation

The implementation of the RTAB-Map is as follows:

- The custom robot URDF made was put in a world using Gazebo environment.
- For visualizing in 3d opened RViz.
- RTAB-Map was initiated.

Teleop script was called for mapping (We used teleop purposely to repeat at places where mapping didn't precisely occur). Also, an autonomous mapping code has been given in repo.

WORKING SECTION

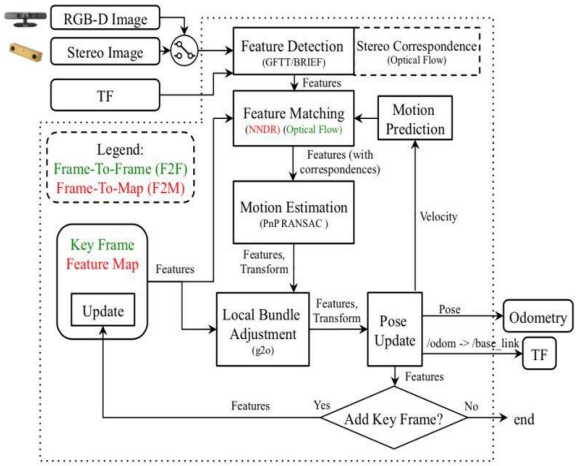
Visual Odometry (VO) is a very vital part of autonomous navigation used with autonomous ground vehicles, aerial vehicles and underwater or surface vehicles. Visual odometry is in layman terms, determining the position information for accurate navigation of the robot using camera image sequences. Usage of encoders has slippage issues which in turn will reduce the accuracy of estimation.

Thus, VO provides a computationally cheaper and accurate odometry results when compared to GPS, wheel odometry, sonar localization or INS.

Visual Odometry via 2 ways:

- Frame2Frame (Registers the new frame against the last keyframe)
- Frame2Map (Registers the new frame against a local map of features created from past keyframes)

F2F is faster as it can be done without descriptors using optical flow directly.



There are 2 approaches for VO- appearance based and feature based. Appearance based VO make use of the pixel intensities to directly estimate the dense correspondences; feature-based VO uses the image intensities to determine the keypoints and matching of the keypoints with corresponding image frame to estimate the ego-location of the robot. We will be evaluating each and every technique along with its classifications in detail and suggest the best fit for Visual Odometry (VO). Our main contributions include:

- The best feature extractors and descriptors combination to be used.
- The most efficient feature matcher and outlier removal technique to be used.
- The most efficient method to estimate the rotational and translational vectors of the camera.

The basic workflow of stereo visual odometry includes Feature Extractor, Feature Matching, Feature Tracking, Outlier removal, Triangulation and PnP for efficient estimation of the pose using Direct Linear Transformation and Levenberg Marquardt Algorithm in its backend. The architecture can be visualized in fig. 1. In this section we will explain all the phases involved in the visual odometry algorithm in detail.

Camera Calibration

Calibration reduces the radial and tangential distortion. Calibration helps to determine the intrinsic and extrinsic parameters along with the distortion coefficients. Intrinsic parameters include focal length and optical centers and extrinsic correspond to the rotational and translational vectors. Implemented using OpenCV library and OpenCV's sample dataset of chess board. First the image points are determined in the chess board as in the first image fig. 1 (a). Then, calibration can be done using the **calibrateCamera()** library. The second image that's fig. 1 (b) shows the undistorted image after calibration is done.

types of pre-processing steps. It for sure will increase computational cost of the system. Its effect on the accuracy of the feature extraction was thus tested precisely.

Feature Extraction

There are a lot of feature extraction methods including FAST, SIFT, ORB, SURF, Shi-Tomasi, BRISK, KAZE, AKAZE, etc. Basic working of all these extractors and comparison with outdoor and indoor environment with different contrasts are shown in this section.

FAST- FAST stands for Features from Accelerated Segment Test [5]. It is one of the fastest feature extraction technique. It is best for live real-time application point of view with efficient computation. The working of FAST is as follows: Corner detection by drawing the bresenham circle around the pixel p and labelling each of the circle from 1-16. Then intensity of N random pixels is compared.

$$S_p = \begin{cases} Ip \rightarrow x \leq Ip - T, \ darker \\ Ip - T \leq Ip \rightarrow x < Ip + T, \ similar \\ Ip + T \leq Ip \rightarrow x, \ brighter \end{cases} \quad (1)$$

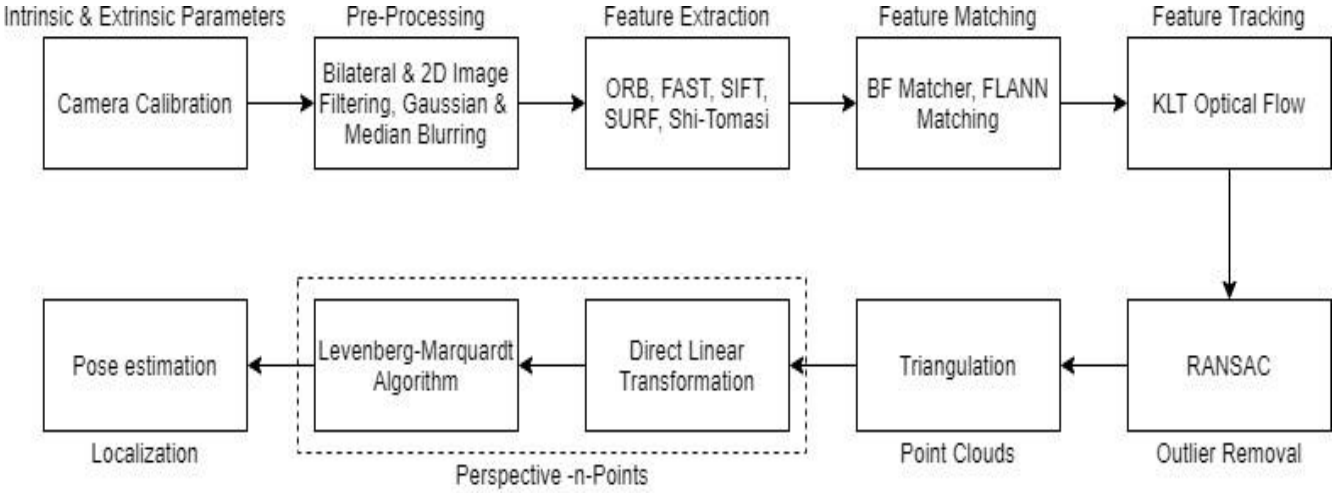


Figure 2. Architecture of the Visual Odometry

Image Enhancement

Smoothening/ blurring- Helps in removing noises in the image leaving most of the image pixels intact. Some of the smoothening methods experimented are image filtering, gaussian blurring, bilateral filtering and median blurring.

Rotation and Warping were also experimented in the pre-processing steps to observe the results with such

Multiple interest points are cast off using the non-maximum suppression which is based on the difference in distance between subsequent keypoints.

SIFT- SIFT stands for Scale-Invariance Feature Transform [6]. Its working is as follows: First interest points are scaled and localized followed by blurring using the gaussian blur operation.

$$L = G\ (x, y, \sigma)\ \times\ (I\ (x, y)) \quad (2)$$

Table 1: Various Feature Extractors

	ORB	SIFT	SURF	KAZE	AKAZE	BRISK
Origin	2011	1999	2006	2012	2013	2011
Scale Invariance	YES	YES	YES	YES	YES	YES
Rotation Invariance	YES	YES	YES	YES	YES	YES
Keypoint Type	FAST	DoG	Hessian	Hessian	Hessian	
Descriptor Type	Binary	Integer	Real	Real	MLDB	Binary
Descriptor Length	32	128	64	128		64

Comparing of the interest points with neighboring octaves is done to determine keypoints based on its extrema values obtained. Rejection of edges and flat region happens next based on intensity measures.

$$Intensity < 0.03 \tag{3}$$

$$HM = \begin{pmatrix} Dxx & Dxy \\ Dxy & Dyy \end{pmatrix}, \frac{Tr\,(HM)}{Det(HM)} = \frac{(R+1)^2}{R} \tag{4}$$

Equation (3) represents rejection criteria for flat regions and (4) for edges. Then the orientation of the keypoints obtained is based on high probable distribution of the orientation of each and every keypoint. Then the descriptors based on orientation and scale of the keypoint is assigned.

SURF- SURF stands for Speeded Up Robust Features [7]. It is a robust and fast technique preferred for its fast computation of operators using box filters, thus enabling real-time applications such as tracking and object recognition. Consists of 2 steps, Feature Extraction and Feature Description. Feature extraction uses integral image, which is a way of calculating the average intensity of pixels in a selected box.

$$Image\ Integral = \sum_{i=0}^{i\leq x}\sum_{j=0}^{j\leq y} I(i,j) \tag{5}$$

Surf uses the Hessian matrix because of its good performance in computation time and accuracy. Descriptors are assigned by first assigning a fixed orientation to each interest point, and then extract feature descriptor.

ORB- ORB stands for Oriented FAST and Rotated Brief [9]. In ORB, oriented FAST algorithm is used to calculate and find the keypoints from the image and rotated BRIEF descriptors are used. Coming to the feature detection part, image pyramidal representation of frames is done for scale invariance and then features from each pyramidal layer is extracted. Firstly, to estimate the orientation, centroid will be found which will be estimated from the moment equations.

$$C = \left(\frac{m10}{m00}, \frac{m01}{m00}\right) \tag{6}$$

Then, orientation patch can be obtained.

$$Orientation = a\ \times \tan 2(m01, m10) \tag{7}$$

Rotated Brief descriptors are used once extraction is done. The orientation patch obtained is used to compute the BRIEF descriptor operation:

$$g\,(p,\theta)\ =\ f(p)\,|\,(x,y)\ \in\ S \tag{8}$$

SHI-TOMASI- Harris corner detector takes the differential of the corner score into account with reference to direction directly. A corner is an important feature in an image, as they are interest points which are invariant to translation, rotation, and illumination. The idea is to consider a small window around each pixel p in an image. Then we measure gradient shift of the central pixel and the pixels around in all 8 directions (using SSD values of pixel value). If shift is above a threshold value, a new feature descriptor is mapped. Small modifications in the Harris Corner detection resulted in the Shi-Tomasi corner detection [10] using the good features to track algorithm.

KAZE- The working of KAZE [13] is as follows: Nonlinear scale-space extrema to detect the features accurately; Nonlinear diffusion filtering with Additive Operator Splitting (AOS) is used instead of gaussian blurring to keep the object boundaries intact.

$$c(x,y,t)\ =\ g(|\,\nabla Io(x,y,t)\,|) \tag{9}$$

Orientation of the feature points are done very similar to SIFT, but instead of aligning in a histogram, points

are used in the vector space. Since the descriptors has to operate at nonlinear scale-space, a modified version of SURF descriptor was used here in KAZE.

AKAZE- AKAZE [12] was built over the KAZE algorithm. It uses an accelerated version of nonlinear scale-space called as Fast Explicit Diffusion (FED). This change was made considering how computationally expensive the AOS process of KAZE was. Also, AKAZE uses a form of local difference binary (LDB) descriptors to increase the computation of the extraction process further.

BRISK- BRISK stands for Binary Robust Invariant Scalable Keypoints [11]. Similar to ORB, BRISK uses pyramidal image representation, wherein stable points are extracted using the adaptive corner detection operator. Unlike other algorithms, BRISK uses binary bitstring [14]. The feature descriptor is generated from this equation:

$$B = \begin{cases} 1 & I(Pj,\sigma j) > I(Pi,\sigma i) \\ 0 & otherwise \end{cases} (Pi, Pj) \in S \qquad (10)$$

Feature Description

BRIEF- BRIEF stands for Binary Robust Independent Elementary Features [8]. It uses binary strings as feature point descriptor. BRIEF identifies the patches around the keypoint and converts it into a binary vector, thereby representing an object. Thus, each keypoint will be described with the help of the binary 1's and 0's. One major consideration about the BRIEF is that it is very noise sensitive as it deals closely with pixel-level images. Thus, smoothening is very important to reduce the noise from the image. Smoothening is done using Gaussian Kernel in BRIEF. Once smoothening is done, the next step is the feature descriptor. Now the problem will come when we will have to calculate the n (x, y), that's the x, y pairs. This (x, y) pairs are called random pair inside the patch. N is the length of the binary vector and τ is the binary test response of the vector. Finding the random pair can be easily done if the length of the binary vector is decided as we can pick from the patch easily then. There are 5 different methods used to calculate the length. They are uniform (GI), gaussian (GII), gaussian (GIII), coarse polar grid (GIV), coarse polar grid (GV). Also, BRIEF relies on less intensity difference between the image patches.

Feature Matching

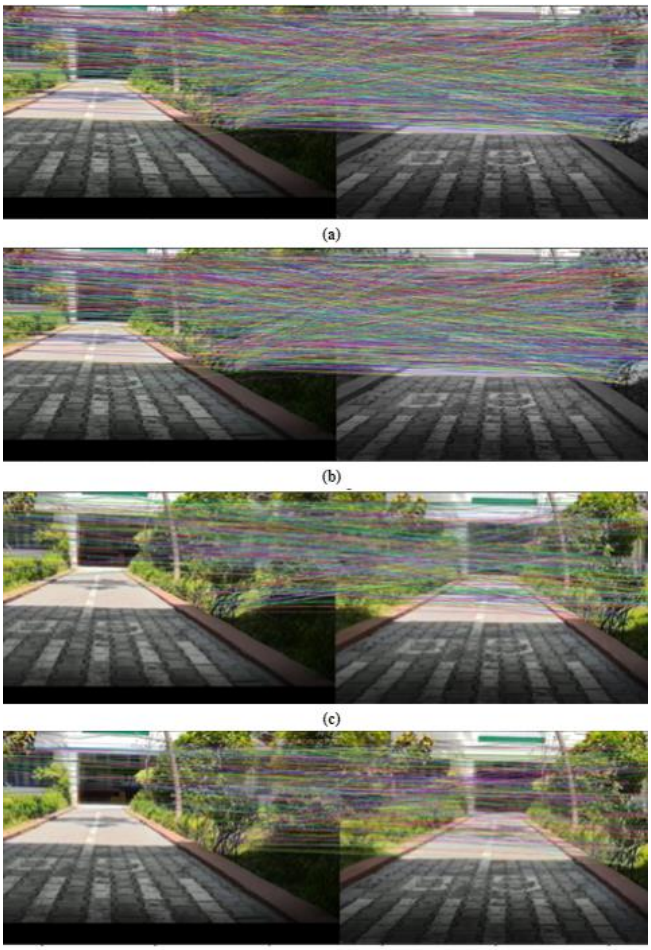
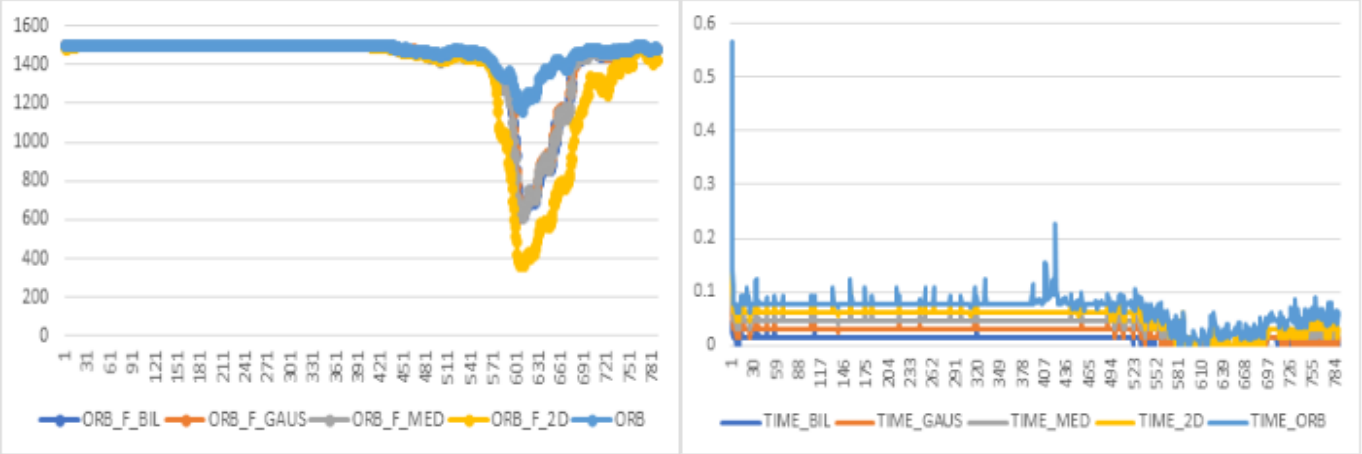


Figure 5: Feature Matching

BF Matcher- It takes the descriptor of each and every feature of first set and matches with all the other features from the second set using distance calculation. It is time consuming and matches all features. There are a lot of distance measurement methods including NORM_L2/ NORM_L1, NORM_HAMMING, etc. Also, Lowes's test is done after matching in general to remove outliers.

FLANN- Stands for Fast Library for Approximate Nearest Neighbor. It has a collection of optimized algorithms to for searching the nearest neighbors in big datasets. It is faster and more accurate than BF Matcher. It has two dictionaries index and search parameters.

Outlier Removal



(a) (b)

Figure 3: Preprocessing

RANSAC stands for Random Sample Consensus. Deals with removing outliers from inliers contained in a data. No real-world sensor readings are perfect. Thus, we use RANSAC which is a simple trial and error method with groups the inliers and outliers separately. Thus, it helps us to throw away the outliers and work with inliers alone which will save our computation and time. It involves a 4-step processing, and they are sampling, scoring, computing and repeating.

Let's assume there are 2D points (Fig. 14 (a)) in which a line has to be fitted. Now we will sample and randomly take one line marked in fig. 14 (b) as our inlier. Now we will compute the number of supporting inliers satisfying the line we drew. This will be done by parallelly extending imaginary lines on both the side of the line assumed to be the inlier with a uniform distance of **δ.** Now each and every point lying inside these imaginary lines constitute to the scoring of the inliers. Now for the fig. 14 (b) there are 4 inliers.

Now we will repeat the steps mentioned above repeatedly and the score will be overwritten with the best score after every iteration. In the Fig. 14 (c), that's after some iterations, we got a line which has 12 points satisfying that line model which is the best score to be obtained. Thus, it will be best fit for the line. This way we can eliminate all the outliers easily.

Rotation and Translation

Table 2. Comparing various Pose Estimation Results

	ORB	SURF	SIFT	KAZE	AKAZE	BRISK
Number of Features	9920257	7880352	9121625	7647275	6038516	11324156
Total Time taken	387.1427	1035.8436	724.0894	4087.2750	922.7840	675.3332
Total % Error	1046.2213	412.6922	241.2476	164.7603	400.5994	678.8387

$$S \times p_c = K \times [R \mid T] \times p_w \quad (1)$$

$$s\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & \gamma & u_0 \\ 0 & f_y & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \quad (2)$$

Some of the assumptions taken will estimating the pose of the robot using stereo vision are:

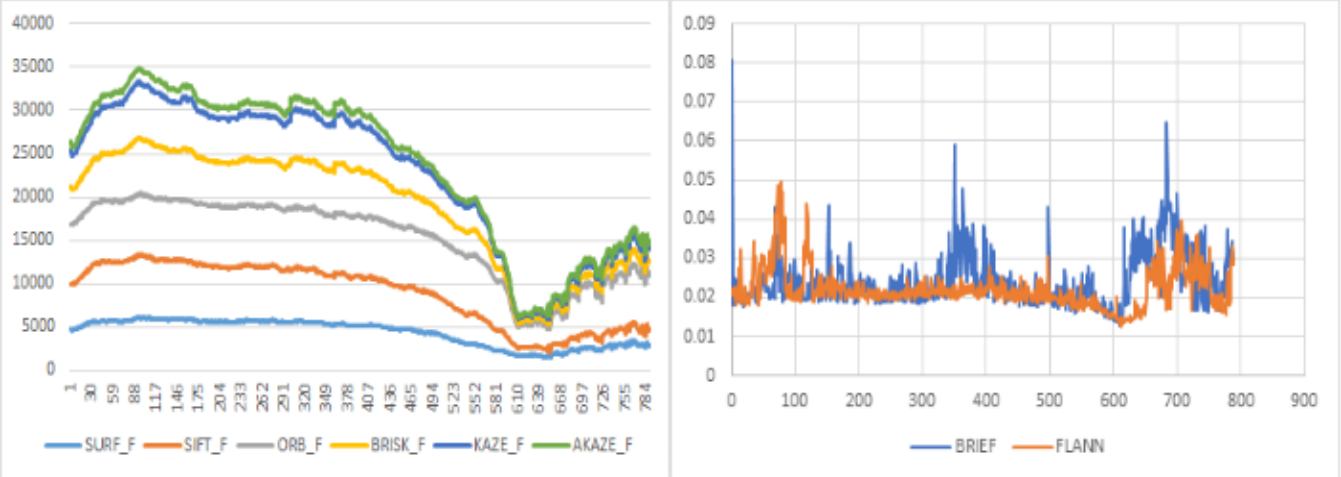
- Camera parameters are known.
- Image co-ordinate is known.

For mono, we will first estimate the estimation matrix derived from the Fundamental matrix. Then we will use recover pose to find the rotation and translational vectors. In stereo, we will use a variant of PnP (that's Perceptive-n-Points). Some of PnP variants include EPnP, MLPnP and the frame-PnP. Basic framework of PnP is as follows:

- Direct Linear Transform (DLT) finds approximate value of Rv and Tv using the projection matrix.

Levenberg-Marquardt Algorithm is a trial-and-error optimization step used to reduce the reprojection error thereby optimizing the estimated Rv and Tv values.

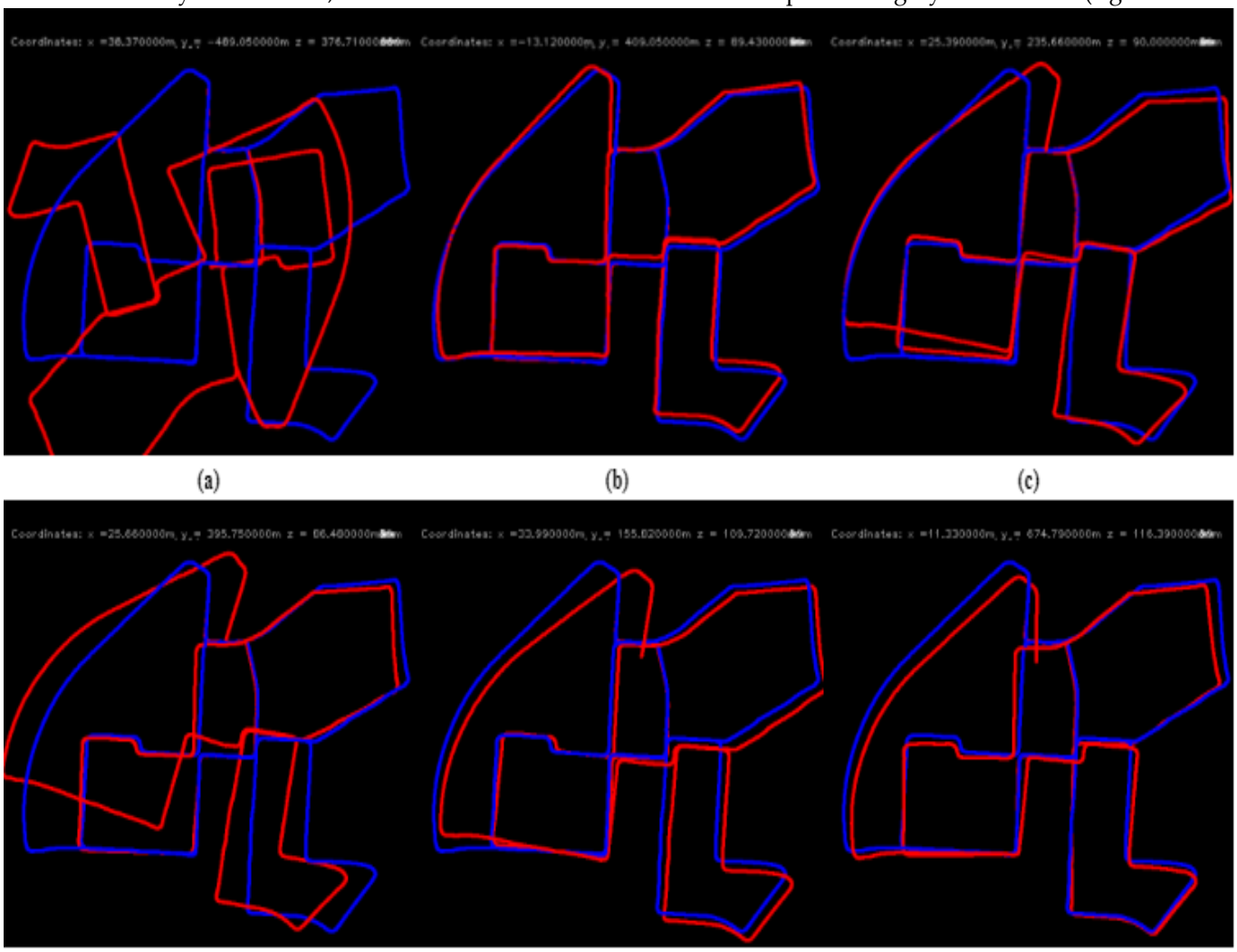
Dataset



(a) (b)

Figure 4: Feature Extractors and Matching techniques

We are using KITTI dataset for comparison and estimating the error/ overlap percentage of our pose estimation algorithm. Some of the specs of KITTI dataset are Grayscale frames, consists of 22 stereo sequences (We have used 1st sequence which has 4500 stereo frames), 11 sequences with ground truth (for finding error/ overlap ratios). We are experimenting with 0th sequence of grayscale frames (right and left



(a) (b) (c)

(d) (e) (f)

Figure 6. Trajectory using various extractors
ORB-SIFT-SURF-AKAZE-KAZE-BRISK

frames). The ground truth trajectory in fig. 7 in red colour is derived from the KITTI data taken using the GPS and IMU readings.

Results and Analysis

Figure 3 (a) shows feature extraction via orb extractors with different pre-processing steps with respect to the number of features and frames whereas fig. 3 (b) shows the time taken for execution of the pre-processing and extraction comparing with various techniques. In figure 4 (a) depicts the number of features extracted using 6 different feature extractors and in figure 4 (b) we check computational time of feature matching techniques including bfmatcher and flann.

When it comes to feature extractors-

- SIFT is most stable,
- SIFT is faster than SURF,
- Shi-Tomasi is fastest,
- Shi-Tomasi extracts less sparse features,
- KAZE is computationally the most expensive algorithm,
- BRISK is the fastest algorithm corresponding to the number of features it works with, thus compromising on the accuracy.

Thus, we conclude to use **SIFT algorithm** for pose estimation/ localization considering its accuracy, drift in trajectory and decent computation with a greater number of features.

When it comes to feature matchers, **FLANN** is more efficient than BF Matcher. For outlier removal, **RANSAC** is preferred over PROSAC and KMeans. For PnP, **Efficient-PnP** or EPnP is preferred. Thus, we are proposing ORB extractors; FLANN matchers; RANSAC outlier removal and EPnP matrix retrieval to be the best techniques that can be used for VO and the results are proved in the previous section.

Thus, a complete visual odometry system has been made strengthening with suitable research and comparisons for the localization of the aerial robot in GPS prone areas.

References

[1]. S. A. K. Tareen and Z. Saleem, "A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK," *2018*

[2]. Bansal, M., Kumar, M. & Kumar, M. 2D object recognition: a comparative analysis of SIFT, SURF and ORB feature descriptors.

[3]. Chien, H.-J., Chuang, C.-C., Chen, C.-Y., & Klette, R. (2016). When to use what feature? SIFT, SURF, ORB, or A-KAZE features for monocular visual odometry.

[4]. Ebrahim Karami and Siva Prasad and Mohamed Shehata Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images

[5]. Rosten, Edward; Drummond, Tom (2006). "Machine Learning for High-speed Corner Detection".[6]. D. G. Lowe, "Object recognition from local scale-invariant features."

[7]. H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features"

[8]. Calonder M, Lepetit V, Strecha C, Fua P. 2010. Brief: binary robust independent elementary features.

[9]. E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF."

[10]. J. Shi and C. Tomasi (1994) Good Features to Track.

[11]. S. Leutenegger et al., "BRISK: Binary robust invariant scalable keypoints,"

All the codes I have written and the work done as an intern in Aero2astro are documented in this repository: https://github.com/jerriebright/VISUAL-ODOMETRY