DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Thesis type (Bachelor's Thesis in Informatics, Master's Thesis in Robotics, . . .)

Thesis title

Author

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Thesis type (Bachelor's Thesis in Informatics, Master's Thesis in Robotics, . . .)

Thesis title

Titel der Abschlussarbeit

Author:	Author
Supervisor:	Supervisor
Advisor:	Advisor
Submission Date:	Submission date

I confirm that this thesis type (bachelor's thesis in informatics, master's thesis in robotics, ...) is my own work and I have documented all sources and material used.

Munich, Submission date Author

Acknowledgments

Abstract

Contents

1 Introduction

1.1 Motivation

In Augmented Reality or Mixed Reality, tracking is one of the most important as well as challenging tasks. Tracking means to detect the position and orientation of an object to a coordinate system in real-time. Tracking has to be very accurate, precise and robust otherwise misalignment will occur between the virtual and real objects in the frame which makes the experience much less pleasant to the user. In the sector of medical augmented reality, the situation can be severe.

There are many methods for tracking an object. We can track an object by multiple camera setup or use mechanical sensors such as Inertial Measurement Unit (IMU) or we can use hybrid approaches combining both camera and mechanical sensors.

The problem with tracking objects with cameras or hybrid systems is that it needs an external camera setup which is not portable and the setup can be problematic to the user. In order to track objects without the camera, we use a mechanical sensor such as IMU, which is portable and embedded in the Head Mounted Display (HMD). Here sensor fusion comes handy. Sensor fusion is a technic for combining sensory data from multiple sensors output which gives better results for tracking. In the case of IMU sensory data, prior work [**8951908**] shows that increasing the number of IMU tends to provide better accuracy.

However recent work is primarily based on data from physical IMUs and multiple cameras in real-life. Some preliminary work has already been carried out with sensor fusion using data from the cameras. They are all used together to predict the object's pose. It seems like a good solution, but the mixing of sensory data with images together in order to train a Deep Neural Network is expensive. Another problem is that the real-life data from this sensor and camera has noise, also they need to calibrate and registration in terms of their mechanical specification. For example, if a mechanical sensor such as Inertial Measurement Unit (IMU) can broadcast data at 240 Hz but a camera can only broadcast at 60 Hz, so in the hybrid system it's difficult to calibrate. To overcome this problem, we aim to use a virtual environment like V-Rep from Coppelia Robotics to collect our data, where we don't need to register the sensors and the data is noise-free. The other benefit of using a virtual environment is that we can use it with very low cost and its less time consuming to run a sensor fusion setup compare to

real-life sensor fusion set up. We like to use this opportunity to try and test a variety of Convolutional Neural Network Structure to predict the pose of a moving object using the data collected from the simulated environment.

1.2 Goals

Our aim is to create a virtual environment when we can place multiple IMU on a object which can move freely withing 6DoF. We aim to create multiple paths for the object in which it can travel and we can run the simulation several times to collect IMU data along with object's pose data. Then we will use the data collected from the environment to train variety of CNN architecture and test their performers in terms of errors displacement of pose compared to the actual object. We will compare the CNN architecture among themselves and find out which architecture works better. The best performing model will be used later in real life scenarios.

1.3 Structure of the Thesis

This document is divided into seven chapters. In the second chapter, the necessary theoretical background for our particular work is presented. In the third chapter, we will briefly discuss the prior work regarding sensor fusion with machine learning, what they did, and how our work is different from them. In chapter 4 we will discuss, how we set up our virtual environment and the data collection methodologies from the environment. Chapter 5 describes the experimental setup for the CNN architecture and the methodologies behind it. We will analyze the results obtained from our CNN in chapter 6, while the conclusions and proposed future work are summarized in chapter 7. Afterward, references and relevant bibliography are presented and the document ends with Appendices where outcomes of every experiment are detailed in their plots.

2 Background

2.1 Sensor

Our modern world us infused with sensors. We use sensors in everyday life concisely or subconsciously. From our smartphone, cars to the state of the art medical equipment, mars roves has a variety of uses of sensors. Although when the term sensor comes up we think about mostly mechanical sensors, such as touchscreen in our smartphone or the fire detection sensor, sensors can be not only mechanical but also chemical, bio, and other variety of types.

These sensors are collecting data every moment in our modern world. For example, our buses, trains, planes, and ships are using GPS to navigate around the world. We are using our smartphone which has a variety of sensors working together like the camera, touch sensors, gyro, GPS, and proximity sensor. They are working continuously in the background and processing data to make our life a little bit easier. Now we can sense and track out heartbeat just using a wristwatch.

The use of smart sensors in our homes, offices, shopping malls, cars is getting bigger and bigger nowadays. All modern means of transport such as cars, buses, trains are using sensors to create a more comfortable transport experience. For instance, proximity sensors for better parking experience, crash prevention. If we talk about air transportation, the impact of sensors is even more important and critical for instance temperature, air pressure sensors. A single failure of any of these sensors could lead to deadly outcomes with fatality. The list goes on when it comes to the use of sensors and their impacts on our life

n the broadest definition, a sensor is a device that can monitor the environment that its designed for. A sensor is a device or an integrated system of devices that senses and responds to certain types of environmental inputs. Light, heat, movement, moisture, pressure, or any of a number of other environmental phenomena could be the specific input. The output is usually a signal converted for reading or continuing processing to a human-readable display at the point of the sensor or electronically transmitted over a network. Due to the different needs, the properties of sensors were changed a lot over time. The need enabled intelligent and intelligent sensors to be created. The advent of intelligent sensors led to the development of microcontrollers used everywhere. The smart sensors have allowed us to interface with other devices and make them work

together as one device.

2.1.1 Sensor Types

As we mentioned before, sensors can be many types, It can be categorized depending on their purpose or depending on how they are made. There are various sensor classifications created by different writers and specialists. An expert in the subject can already use the following sensor classification, but it is an extremely simple sensor classification. They are divided into active and passive in the first classification of the sensors. Active sensors are those that require a power signal or an external excitation signal. On the other hand, passive sensors do not require an external power signal and generate an output response directly. Analog and optical sensors are the final grouping of the sensors. Analog capabilities generate an analog output.

We have a long list of sensors, which we use in our everyday work. If we want to speak briefly about them, we can hardly cover each sensor. We will discuss a few of them which is somewhat related to our work. The sensor mention below are regardless of their classes, they are a mix of digital, analog, active sensors.

2.1.1.1 Temperature sensors or thermal sensors

This system collects temperature information from a system and transforms it into a way that another system or individual can understand. Mercury in a glass thermometer is the simplest example of a temperature sensor. Mercury expands in the glass and contracts to rely on temperature changes. The external temperature is the source factor for measuring the temperature. In order to calculate the temperature, the spectator measures the location of mercury. Two basic temperature sensors are available:

- Contact sensors – This type of sensor requires physical direct contact with the sensed object or media. They monitor solids, liquids, and gas temperatures across a broad range of temperatures.
- Non-contact Sensors – No physical contact with the object or the media is required for this type of sensor. They monitor non-reflective solids and liquids, but because of natural transparency, they are not useful for gasses. These sensors use the Law of Plank for temperature calculation. This law covers heat radiated from the heat source for temperature measurements.

2.1.1.2 Sonar sensors

Sonar sensor: As the name suggests, the Sonar sensor also known as Ultrasonic sensor works based on an ultrasonic pulse time calculation to measure distance.

Figure 2.1: A typical temperature sensor.

Source: emerson.com

To measure the specific distance from the sensor, this can be calculated based on this formula [**sonar1**]:

Distance = 1/2 T x C

where T is time and C is the spped of sound in specific temparatuer of earth environment. At 20° C (68° F), the speed of sound is 343 meters/second (1125 feet/second), but this varies depending on temperature and humidity.

2.1.1.3 IR Sensors

These types of sensors sense the infrared light. In general, all objects on the infrared (IR) spectrum emit thermal radiation. That kind of radiation that is not visible to the human eye is measured by the infrared sensor. The biggest advantage of the IR sensor is that it uses very cheap and available and very easy to use. One of the major drawbacks of these sensors is that it pruned to background noise.

The working of IR sensor and sonar sensor is similar, where the sonar sensor emits soundwave and have a receiver for the sound wave the IR use infrared light instead of sound.

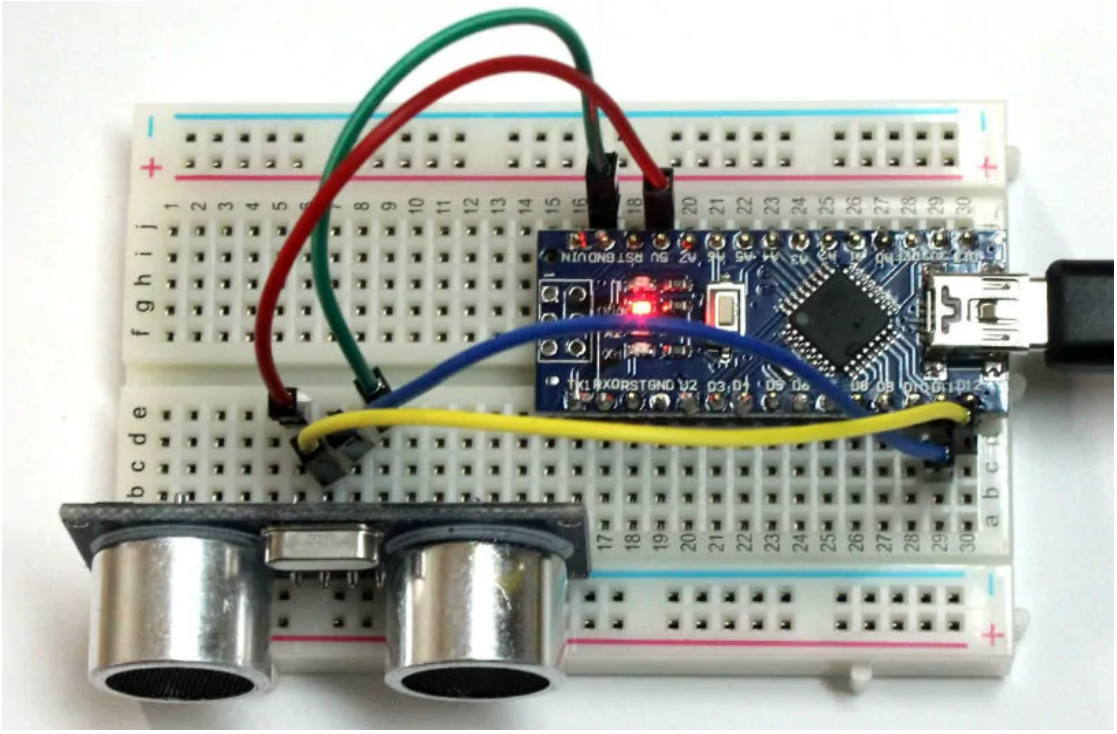


Figure 2.2: A sonar sensor combined with microcontroller

Source: arrow.com

2.1.1.4 UV Sensors

Such sensors measure the intensity or strength of the UV radiation incident. It has a longer wavelength than x-rays but is also shorter than the visible radiation. For robust ultraviolet sensing, an active material called polycrystalline diamond is used. UV sensors can detect exposure to ultraviolet radiation from the environment.

The UV sensor recognizes one energy signal type and transmits different energy signals. They are directed to an electric meter to observe and record these output signals. The output signals are transmitted for the generation of graphs and reports to a software-based computer through an analog-to-digital converter (ADC).

2.1.1.5 Touch Sensor

A touch sensor functions as a variable resistor according to the position of the touch. As seen in figure **??**

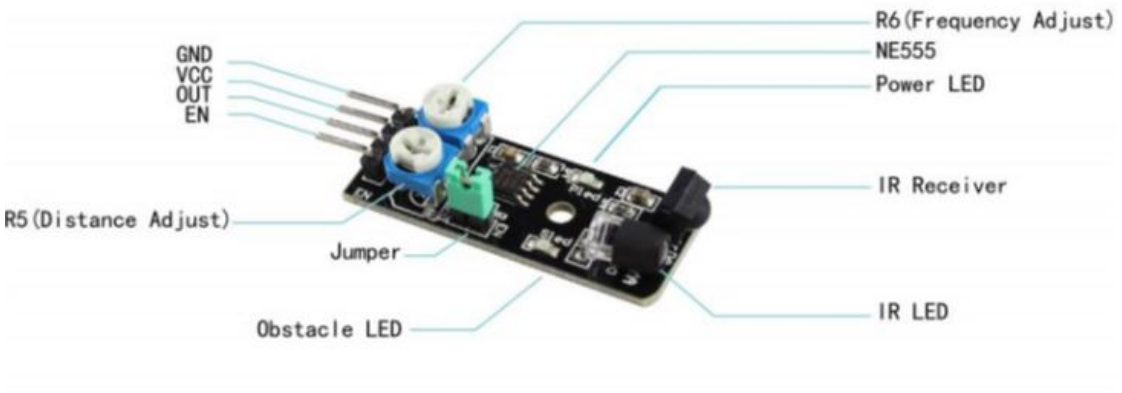


Figure 2.3: An IR sensor architecture

Source: ram-e-shop.com

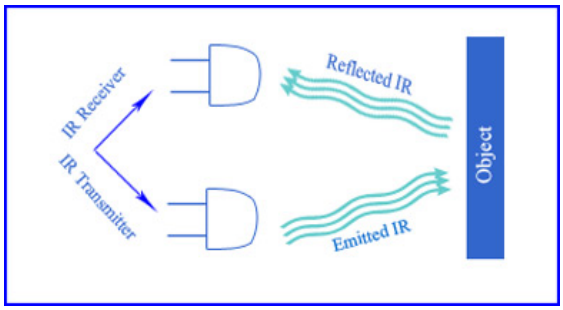


Figure 2.4: Working principal of an IR sensor

Source: engineersgarage.com

A touch sensor made of a substance that is absolutely conductive such as copper Isolated material for the spacing of foam or plastic or material which is partly conductive. The way the touch sensor works is that the surface is partly conductive and opposes the present flow. The key concept of the linear position sensor is that when the length of this material, to be traveled by the current, is greater, the current flow is contrasted. The resistance of the material is therefore varied by adjusting the location at which it comprises the substance that is completely conductive.

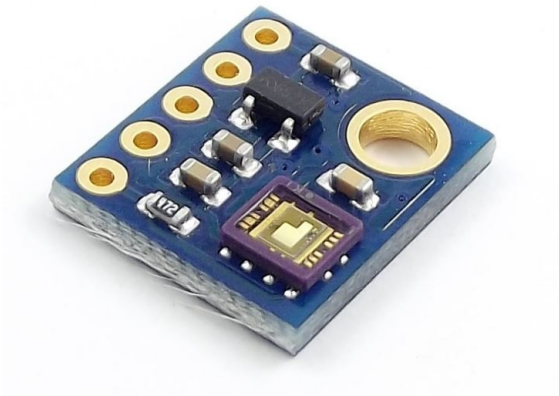


Figure 2.5: UV sensor

Source: roboticsbd.com

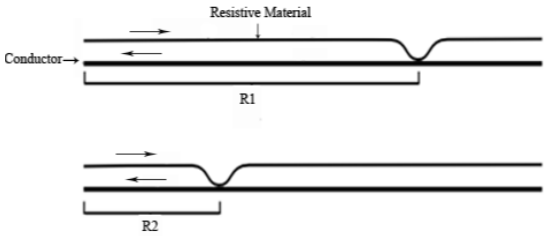


Figure 2.6: Working mecanism of a touch sensor

2.1.1.6 Global Positioning System (GPS) sensor

We use the GPS sensor almost every day in our life, mostly via our phone or in our car, the GPS sensor is used to navigate around the world.

A minimum of 24 operational satellites orbit over 12,000 kilometers above ground at any given time. The locations of the satellites will always include up to 12 satellites in the sky above your location. The primary function of the 12 visible satellites is to relay radio frequency (1.1-1.5 GHz) information back to earth. A soil-based receiver or GPS module may measure its location and time with this information and other mathematical details. [**misra2006globa**]

2.1.1.7 LiDAR

Lidar is an analogy for "light detection and ranging." The device uses eye-safe laser beams to produce a 3D view of the surroundings being scanned.

Figure 2.7: A Lidar device used for automotive

Source: Automotive Lidar Sensor Market 2017 - Leddar, Quanergy

A typical lidar sensor absorbs the ambient atmosphere with pulsed light waves. Such pulses rebound and come back to the sensor. To calculate the distances traveled, the sensor takes time for every pulse to return to the sensor. This procedure is repeated millions of times per second and creates a precise 3D environmental map in real-time. This chart can be used for secure surfing by an onboard computer. Lidar is mostly used for advance autonomous systems to measure its surrounding environment.

2.1.1.8 Gyroscope sensor

A device that senses angular velocity is gyro sensors, also known as angular rate sensors or angular velocity sensors. Gyroscopes in consumer électronics were not only found in compasses, ships, computer pointing instruments, etc.

Since the gyroscope allows orientation and rotation to be measured, designers have combined them with modern technology. The gyroscope technology has made it possible to detect the movement inside a 3D world more reliably than the previous lone accelerometer on most smartphones. Gyroscopes are also paired with accelerometers for better direction and motion sensing in consumer electronics. The gyroscopes inside smartphones don't have wheels and gimbals such as the conventional mechanical ones in an old aircraft, they are MEMS (Micro-Electro-Mechanical Systems) gyroscopes instead.

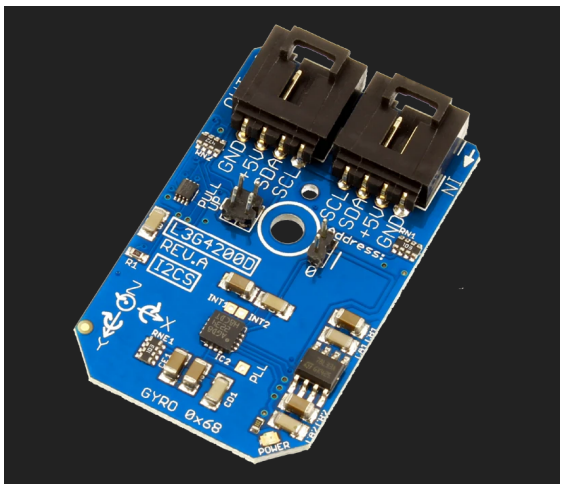


Figure 2.8: An MMSE gyroscope

Source: controleverything.com

2.1.1.9 Accelerometer sensor

It can be used for the measurement of the acceleration exerted on the sensor, as its name implies. The acceleration is usually performed in two or three components of the axis-vector, which make up the acceleration. [**FERNANDEZ20138**] There are a few uses of Accelerometers, remote controls for videos, smartphones, etc. Accelerometer usually combined with a Gyroscope for better sensing. Accelerometers give us two data types: Static force used on the sensor due to gravity detection in the direction of orientation and Sensor strength, acceleration to motion, force detection

2.1.1.10 Smart Sensors

Due to the different needs, the sensor characteristics were adjusted a lot over time. The needs allowed intelligent and intelligent sensors to be developed. The IEEE 1451 standard defines smart sensors as a physical device or a combination of sensors with physical connectors for communication with processor and data network. [**5739775**].

Fig (smartSensorStructure) source cite [5739775]

The advent of smart sensors contributed to the development of microcontrollers used all over the world. Because of the intelligent sensors, interfacing and working with other peripherals as a single device was possible. These smart sensors allow us to reduce data volume for communication with the main processor and made faster communication possible. It also reduced power consumption because multiple sensors

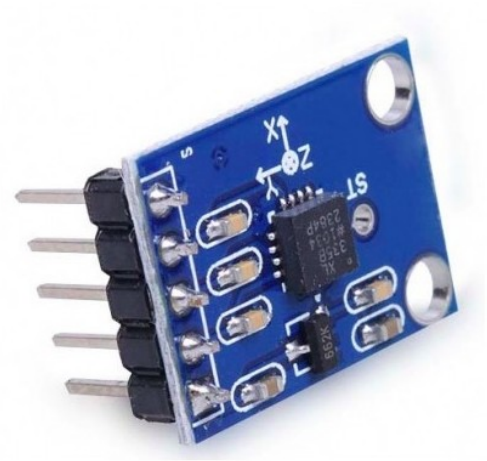


Figure 2.9: An electronic accelerometer sensor

Source: nuttyengineer.com

are used together that use the same processing unit. The major benefit of these smart sensor is that they are typically small and very easy to integrate with other systems even with other sensors.

Here come the uses of IMU (Inertial Measurement Units). In the next section, we describe the properties of an IMU.

2.1.2 IMU (Inertial Measurement Units)

As our goal is to use IMU as our main sensor to collect data for acquiring objects position and orientation, this is our principal sensor to study. An IMU is typically a combination (Accelerometer, Gyroscope, Magnetometer) of sensors working together. In this chapter, we will discuss the IMU, how its functions, its general structure, and the use cases.

2.1.2.1 General structure

The IMU typically consists of 3 individual sensors Accelerometer, Gyroscope, and a Magnetometer. Sometimes it comes with only an Accelerometer and Gyroscope. The Accelerometer, Gyroscope, and a Magnetometer provides the data of acceleration and velocity, orientation and angular velocity, gravitational forces respectively. This

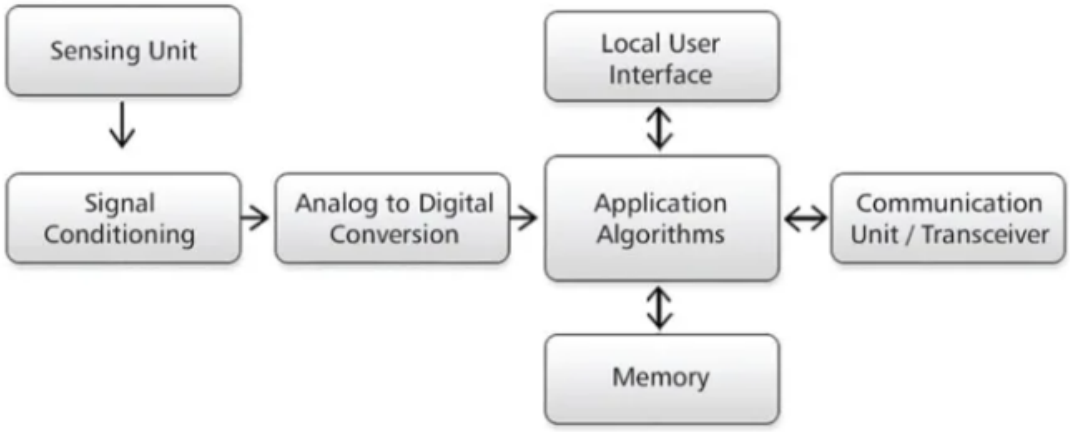


Figure 2.10: Smart sensor building blocks

Source: Premier Farnell Ltd

typically gives us the 9 axes of measurement on the other hand the IMU consist of only Accelerometer and Gyroscope provide 6 axes of measurement. Early days the size of IMU was considerably larger compared to the current IMUs. Its because those are made mechanically but thanks to the MEMS or Micro-Electro-Mechanical Systems that are made possible to make these sensors electronically it's miniaturized the size considerably. [**MEMS**]. This helps us to integrate with other microcontrollers or microprocessors like Arduino or Raspberry Pi. Due to its miniature of size sometimes the IMUs are also known as MIMU (Miniature Inertial Measurement Units) [**2018233**]

2.1.2.2 Working principal of IMU

The main purpose of IMU is to measure and objects pose relative to its inertial space. Acceleration and angular speed are based on inertial principles are measured.

Accelerometers calculate in a particular direction linear acceleration. An accelerometer can be also used as a downward force to measure gravity. Integrating acceleration once shows a velocity estimate, and integrating again provides you with a position estimate.

$$f^b = R^{bn} \left(a_{ii}^n - g^n \right) \tag{2.1}$$

Although accelerometers can measure linear acceleration, they are unable to measure orientation or rotational motion. Nevertheless, gyroscopes measure angular velocities

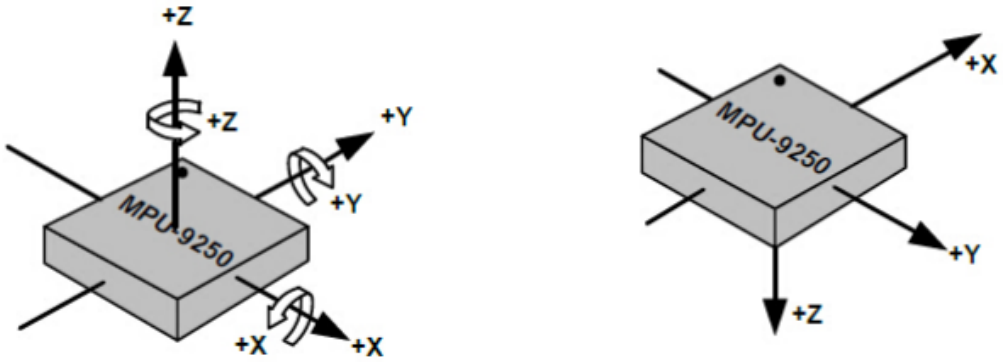


Figure 2.11: An IMU with 6DoF workiks with a magnetometer for 9DoF

Source: Stanford.edu lecture 9 of EE267: Virtual Reality S2020

along three axes: pitch (x-axis), roll (y-axis), and yaw (z-axis). While a gyroscope does not have an initial frame of reference (like gravity), for measuring angular position, you can combine its data with data from an accelerometer. Gyroscope data output angular velocity (deg/s).

The gyroscope measures the angular velocity of the body frame with respect to the inertial frame, expressed in the body frame [**ROS**], denoted by ω. This angular velocity can be expressed as

$$\tilde{\omega} = \omega + b + \eta \tag{2.2}$$

The estimated orientation can be obtained by integrating these angular velocities. We can also use Taylor series to measure discrete-time approximation of integration

Using 3 Gyro and Accelerometer we can obtain an estimated pose of the object in 3D space. The working principle of a single unit of IMU can be seen in the figure 2.12

2.2 Sensor Fusion

Sensor fusion is the act of combining data acquired from two or more sensors sources such that the resulting combination of sensory information provides a more certain description of factors observed by the separate sensors than would be if used individ ually [Elmenreich, 2001]. Sensor fusion is pertinent in many applications that entail

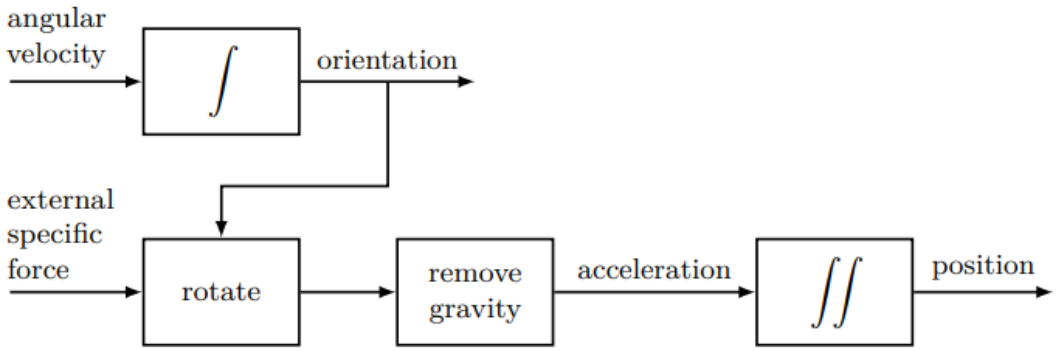


Figure 2.12: working principle of a single unit of IMU

Source: Stanford.edu lecture 9 of EE267: Virtual Reality S2020

the use of multiple sensors for inference and control. Examples of applications in clude intelligent and automated systems such as automotive driver assistance systems, autonomous robotics, and manufacturing robotics [Elmenreich, 2007]. Sensor fusion methods aim to solve many of the problems inherently present in sensors. Several important benefits may be derived from sensor fusion systems over single or disparate sensor sources. The benefits of sensor fusion over single source are the following [Elmenreich, 2001]:

- Reliability: Using multiple sensor sources introduces more resilience to partial sensor failure, which leads to greater redundancy and reliability.
- Extended spatial coverage: Each sensor may cover different areas. Combining the covered areas will lead to a greater overall coverage of surrounding environment and accommodate sensor deprivation.
- Extended temporal coverage: Each sensor may update at different time intervals, and thus interpolated sensor updates can be joined for increased tem poral coverage and decreased sensor deprivation.
- Increased Confidence: Combining sensor data will provide increased confidence by providing measurements resilient to the uncertainties in any particular sensor based on the combined coverage and error mitigation of all sensors.
- Reduced Uncertainty: Given the resilience of multiple sensors to the specific uncertainty of any one, the overall uncertainty of the perception system can be drastically reduced using sensor fusion.

- Robustness against noise: Multiple sensor sources can be used to determine when any one sensor has encountered noise in order to mitigate influence of noise in the system.
- Increased Resolution: Multiple sensor sources can be used to increase the resolution of measurements by combining all observations.

According to [Rao, 2001], sensor fusion can yield results that outperform the measurements of the single best sensor in the system if the fusion function class satisfies a proposed isolation property based on independently and identically distributed (iid) samples. The fusion function classes outlined that satisfy this isolation property are linear combinations, certain potential functions and feed-forward piecewise linear neural networks based on minimization of empirical error per sample.

2.3 Current Sensor Fusion Systems

Many researchers and companies have developed their own versions of sensor fusion systems for various purposes. Many of these systems are well-known an widely used in practice within the fields of automotive and robotics. In [Steux et al., 2002], a vehicle detection and tracking system using monocular color vision and radar data fusion using a 3-layer belief network was proposed called FADE. The fusion system focused on lower-level fusion and combined 12 different features to generate target position proposals each step and for each target. FADE performed in real-time and yielded good detection results in most cases according to scenarios recorded in a real car. A fusion system for collision warning using a single camera and radar was applied to detect and track vehicles in [Srinivasa et al., 2003]. The detections were fused using a probabilistic framework in order to compute reliable vehicle depth and azimuth angles. Their system clustered object detections into meta-tracks for each object and fused object tracks between the sensors. They found that the radar had many false positives due to multiple detections on large vehicles, structures, roadway signs and overhead structures. They also found that the camera had false positive detections on larger vehicles and roadway noise like potholes. Their system worked appropriately for nearby vehicles that were clearly visible by both sensors, but the system failed to detect vehicles more than 100 meters away due to insufficient resolution or vehicle occlusion. In [Dagan et al., 2004], engineers from Mobileye successfully applied a camera system to compute the time to collision (TTC) course from size and position of vehicles in the image. Although they did not test this theory, they mentioned the future use of radar and camera in a sensor fusion system since the radar would give more accurate range and range-rate measurements while the vision would solve

angular accuracy problems of the radar. When the research was conducted, it was suggested that the fusion solution between radar and camera was costly, but since then, costs have decreased. A collision mitigation fusion system using a laser-scanner and stereo-vision was constructed and tested in [Labayrade et al., 2005]. The combination of the compli mentary laser scanner and stereo-vision sensors provided a high detection rate, low false alarm rate, and a system reactive to many obstacle occurrences. They men tioned that the laser-scanner was fast and accurate but could not be used alone due to many false alarms from collisions with the road surface and false detections with laser passes over obstacles. They also mentioned that stereo-vision was useful for modeling road geometry and obstacle detection, but it was not accurate for comput ing precise velocities or TTC for collision mitigation. In [Laneurit et al., 2003], a Kalman filter was successfully developed and applied for the purpose of sensor fusion between multiple sensors including GPS, wheel angle sensor, camera and LiDAR. They showed that this system was useful for detection and localization of vehicles on the road, especially when using the wheel angle sensor for detecting changes in vehicle direction. Their results revealed that cooperation between the positioning sensors for obstacle detection and location paired with LiDAR were able to improve global positioning of vehicles. A deep learning framework for signal estimation and classification applicable for mobile devices was created and tested in [Yao et al., 2016]. This framework applied convolutional and recurrent layers for regression and classification mobile comput ing tasks. The framework exploited local interactions of different sensing modalities using convolutional neural network (CNN)s, merged them into a global interaction and extracted temporal relationships via stacked GRU or LSTM layers. Their frame work achieved a notable mean absolute error on vehicle tracking regression tasks as compared to existing sensor fusion systems and high accuracy on human activity recognition classification tasks while it remained efficient enough to use on mobile devices like the Google Nexus 5 and Intel Edison. A multimodal, multi-stream deep learning framework designed to tackle the ego centric activity recognition using data fusion was proposed in [Song et al., 2016b]. To begin, they extended a multi-stream CNN to learn spatial and temporal features from egocentric videos. Then, they proposed a multi-stream LSTM architecture to learn features from multiple sensor streams including accelerometer and gyroscope. Third, they proposed a two-level fusion technique using SoftMax classification layers and dif ferent pooling methods to fuse the results of the neural networks in order to classify egocentric activities. The system performed worse than a hand-crafted multi-modal Fisher vector, but it was noted that hand-crafted features tended to perform better on smaller datasets. In review of the research, it seems there were limited amounts of data, flaws in the fusion design with SoftMax combination and flaws in the sensors, such as limited sensing capabilities. These factors all may have led to worse results than hand-crafted features on the utilized dataset. In

[Wu et al., 2015], a multi-stream deep fusion neural network system using con volution neural networks and LSTM layers was applied to classify multi-modal temporal stream information in videos. Their adaptive multi-stream fusion system achieved an accuracy level much higher than other methods of fusion including averaging, ker nel averaging, multiple kernel learning (MKL), and logistic regression fusion methods.

2.4 Machine Learning

Machine Learning is a term that describes algorithms that learn from data. Goodfellow et al. quotes Mitchell from 1996 for a definition of what learning means, A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E. There is a wide variety of tasks T that can be done. For instance common tasks are regression, classification, anomaly detection, denoising and translation. Depending on what experience E they see during the learning process the most algorithms can be classified as supervised or unsupervised learning. Supervised learning algorithms experience a data set together with target values or labels that act as instructions on what to do. The unsupervised learning algorithms don't have any labels or targets but instead try to learn about the structure of the data set. The performance P can be how far off from the labels the output of the algorithm is in the case of supervised learning. This performance measure is often difficult to choose.

2.5 Deep Learning

Deep Learning is a term used for training deep artificial neural networks, or ANN s. An ANN is a function approximator consisting of multiple functions,

also called neurons. An ANN is called a feedforward ANN if there are no feedback connections of the output of the ANN being sent to itself. Each neuron is a linear function of some inputs, fed into a nonlinear function, called activation function. If the input of a neuron is a vector x, the output of the neuron gi can be written as

2.5.1 Convulitional Neural Network

The non-linear functions called activation functions, such as K in Equation 2.1, are typically fixed non-linear functions that are used to make the ANN able to approximate non-linear functions. If the non-linear activation functions weren't used, the output of the ANN would still be a linear function of the inputs x. In deep learning, there are a

few commonly used functions that have become standard as activation functions. The rectifying linear unit, or ReLU is a function that is defined as

2.5.2 Back-propagation

When training a ANN to approximate a function, gradient based optimization is commonly used. To compute the gradient of a (loss) function f with respect to the parameters an algorithm called back-propagation is commonly used. It back-propagates from the objective function to gradients of the different weights and biases in the ANN to compute the gradient of the objective function with respect to all the parameters. If we have a function can be a cost function in a supervised learning setting but could also be other functions like a reward function in the reinforcement learning setting that will

2.5.3 Gradient based optimization for deep learning

The ANN weights and biases are updated to optimize some objective function with gradient based optimization, using the gradients computed with the backprop algorithm. The classic algorithm for optimizing the parameters in a ANN is called Gradient Descent. be an objective function that we want to minimize, where x is the input to the ANN the trainable parameters of the ANN. Then moving in the opposite direction of the gradient of L with respect to we move the parameters in a direction that makes the objective function smaller, which is what we want. However, usually x are random samples and the true gradient is the expected value of the gradient with respect to the random samples actually used. Thus when computing the gradient of the loss function as a function of some samples x, we are computing an unbiased noisy estimate of the gradient. This is referred to as Stochastic Gradient Descent, or SGD. Stochastic Gradient

2.5.4 Batch Normalization

Batch Normalization is a recent method in deep learning used to be able to train networks faster and use higher learning rates with decreased risk of divergence. It does so by making the normalization a part of the model architecture, fixing the mean and variances of the inputs to a layer by a normalization step. This makes the risk of the inputs to the activation functions getting in a range where the gradient vanishes smaller and allows for the use of higher learning rates.

2.6 Deep Learning

Deep Learning is a term used for training deep artificial neural networks, or ANN s. An ANN is a function approximator consisting of multiple functions,

also called neurons. An ANN is called a feedforward ANN if there are no feedback connections of the output of the ANN being sent to itself. Each neuron is a linear function of some inputs, fed into a nonlinear function, called activation function. If the input of a neuron is a vector x, the output of the neuron gi can be written as

2.6.1 Activation Functions

The non-linear functions called activation functions, such as K in Equation 2.1, are typically fixed non-linear functions that are used to make the ANN able to approximate non-linear functions. If the non-linear activation functions weren't used, the output of the ANN would still be a linear function of the inputs x. In deep learning, there are a few commonly used functions that have become standard as activation functions. The rectifying linear unit, or ReLU is a function that is defined as

2.6.2 Cost Functions

To train a ANN, cost functions are usually minimized. In supervised learning, where we have labeled data to learn from, the cost functions are more straightforward than for instance in reinforcement learning that we will discuss in section The two main problems of supervised learning are regression and classification. In regression one wants to predict a numerical value whereas in classification the goal is to predict which class something belongs to given some inputs. Different loss functions are common for regression and classification. Most of them are however derived from the same principle, the one of Maximum Likelihood.

2.6.3 Back-propagation

When training a ANN to approximate a function, gradient based optimization is commonly used. To compute the gradient of a (loss) function f with respect to the parameters an algorithm called back-propagation is commonly used. It back-propagates from the objective function to gradients of the different weights and biases in the ANN to compute the gradient of the objective function with respect to all the parameters. If we have a function can be a cost function in a supervised learning setting but could also be other functions like a reward function in the reinforcement learning setting that will

2.6.4 Gradient based optimization for deep learning

The ANN weights and biases are updated to optimize some objective function with gradient based optimization, using the gradients computed with the backprop algorithm. The classic algorithm for optimizing the parameters in a ANN is called Gradient Descent. be an objective function that we want to minimize, where x is the input to the ANN the trainable parameters of the ANN. Then moving in the opposite direction of the gradient of L with respect to we move the parameters in a direction that makes the objective function smaller, which is what we want. However, usually x are random samples and the true gradient is the expected value of the gradient with respect to the random samples actually used. Thus when computing the gradient of the loss function as a function of some samples x, we are computing an unbiased noisy estimate of the gradient. This is referred to as Stochastic Gradient Descent, or SGD. Stochastic Gradient

3 Relatedwork

4 Simulation Setup

The Experiment Setup can be divided into two phases, one is set up the simulation and collect data from the simulated setup. We call title this two-section as Simulation and Data Collection respectively. In this section we describe the detailed information for the simulation software and how we gathered data from the simulation and preprocessed the data to feed in the CNN algorithm.

We use simulated data because its easier to deal without any noise and doesn't need calibration and registration. The other advantages of the simulated data are that the simulated environment is ideal for this kind of experiment without any noisy data and its ideal for the analysis of the result before using it into the real world scenario and compare with it.

We obtained the from the simulation as .txt formate. then we processed the data and convert the data points in csv formate to use in our CNN model.

4.1 Simulation

As for the simulation we need to choose a simulation software that fulfills our requirement for simulating IMUs and moving the sensors on a flat plane on a predefined path. The other crucial requirements were, the data must be faultless and noise-free. For example, the item that had been utilized to mount the IMUs needed to move openly satisfying Six Degrees of Freedom (6DoF).

Moreover, the estimations from the virtual IMUs must be as exact as conceivable to imitate a real situation. We had fewer options for the simulation software to choose from. Among the simulation software we checked for our simulation MATLAB and CoppeliaSim Robotics were better performing.

4.1.1 Simulation software

After intensive investigation, we chose to go with the CoppeliaSim Robotics Education version for our simulation tool. Beforehand this product was known as V-Rep by Coppelia Robotics. One of the primary purposes behind picking this simulation tool is the assortment of accessible simple to utilize alternatives to mimic genuine situations. This product is created to recreate genuine situations for various mechanical parts for

example Robot arms, Hexapods. It additionally has a wide scope of different segments for reproduction purposes. For instance – Infrastructure, furniture, family unit things, office things, and even people for reenactment purposes. This product offers a scope of usable virtual sensors for example – accelerometers, spinners, vision sensors, laser scanner, GPS sensors, and so forth.

The training adaptation is free for all. Since we didn't need to stress over the frequencies of various IMUs since every one of them is reproduced, the CPU recurrence made a difference the most while gathering the information. To analyze the reason, we ran the recreation on various CPU load. For example, with 4 IMUs and no other application running on the foundation, it took roughly 73 minutes to gather 1,06,437 information focuses. In a similar arrangement, however, with Google Chrome running on the foundation, the reenactment could just gather 47,509 information focuses on a similar time. It is required to run the reproduction remain solitary to get around the comparative number of information that focuses on a comparable time period.

In the following section we will describe the the components of the CoppeliaSim Robotics and how we used it to create the simulation

4.1.2 Scene Object

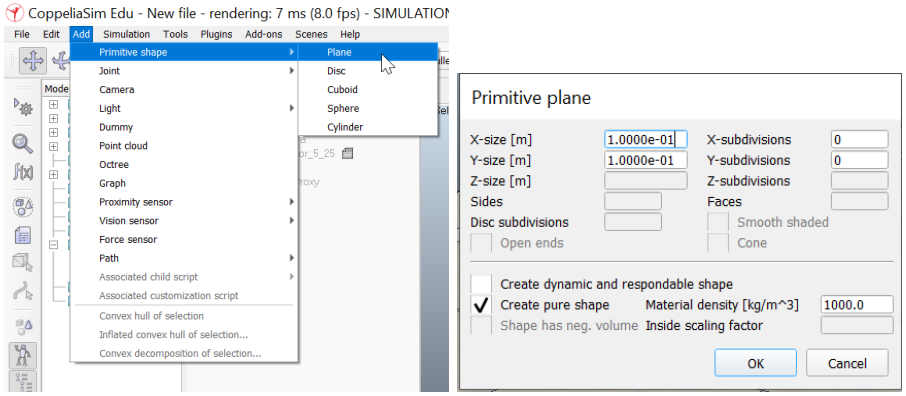
To recreate a real-world like situation, the main thing we required was an object for the scene. The object would represent the qualities of a moving segment in the simulated scenario

For our investigation, we required an object to contain the IMUs that would move along on a pre-characterized way. Obviously, 6 DoF must be guaranteed to make however much as could reasonably be expected accurate data. For our first methodology, we utilized effectively accessible predefined shapes that built-in the software. The software gives a number of predefined shapes such as plane, circle, cuboid, circle, and cylinder. We picked to utilize a two-dimensional plane as our object since it is the nearest to our model scenario.

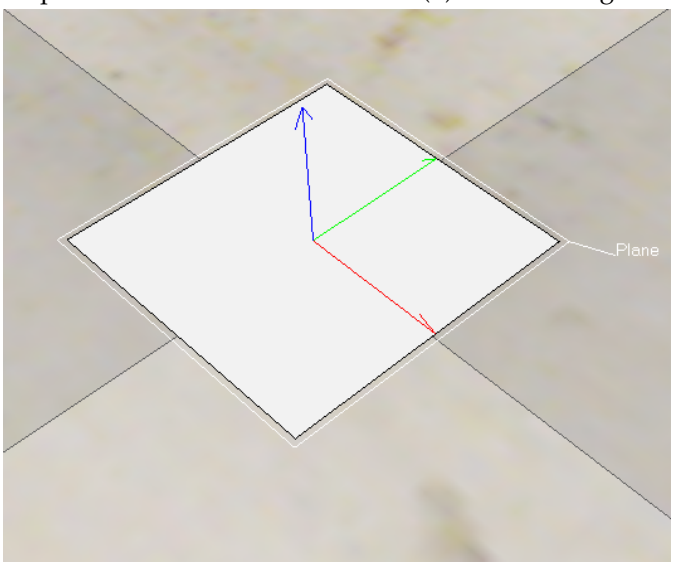
The plane is sufficiently large to contain at least 16 IMUs which is the most noteworthy number of IMUs we chose to utilize. The examples for putting the IMUs on the plane had been chosen together which would in the end help us to get the most ideal and solid information.

To add our ideal object to the scene, the following steps were :

1. Click on 'Add' on the menu bar.
2. Go to the first option; 'Primitive Shape'.
3. Click on 'Plane' to add a two dimensional plane in our scene.



(a) 2D plane to the scene. (b) Plane Configaration.



(c) Created 2D plane.

Figure 4.1: Creating scene object.

4. No change needed in the 'Primitive plane' dialog box and click 'OK'.

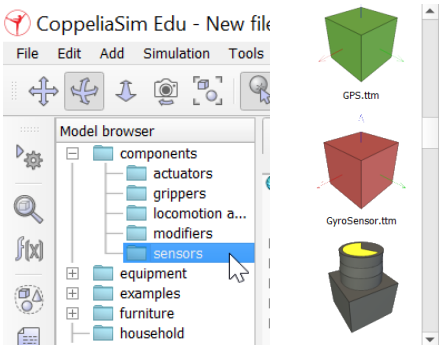
4.2 Inertial Measurement Units

The next task is to add IMUs (Inertial Measurement Units) into our plane. The IMU containes an accelerometer and a gyroscope sensor. In the sensor section of the simulation tool, there is no IMU sensor out of the box but there are an accelerometer and a gyroscope sensor. To create IMU we combined these 2 sensors and added to to

our plane.

To add one of those sensors to our scene, the following steps are needed to be followed:

1. Click on 'components' on the 'Model browser' pane.
2. Click on 'sensors'; this would open a new pane for all the available sensors just below the browser pane.
3. Scroll down the pane to select our desired sensors (accelerometer, gyroscope).
4. Drag and drop the sensors on our plane in the scene.



(a) selection sensor componant.

(b) selction accelerometer and a gyroscope sensor.

Figure 4.2: Creating IMU.

To combine the accelerometer and a gyroscope sensor and make a single IMU we need to change the LUA script attached with every sensor. Although it was possible just to drag and drop the sensors but we wanted to create a custom script to add a single unit IMU to incorporate our custom simulation control panel. we will discuss the control panel later on in the upcoming section.

4.2.1 Making IMU

We created a custom IMU combining gyroscope and accelerometer in a separate scene. To do that we selected the gyrosensor script as our base script. Then we added the

accelerometer into the scene with the gyrosensor. Both of the components Lua scripts were auto-generated which can sense and collect data by the software. We changed the Lua script for both of the gyrosensor and accelerometer to get the sensor data out of the software and saved in a .txt file. When we are satisfied with our work, we saved the whole script as a new sensor names IMU. This IMU now consists of one accelerometer and one gyroscope also we can add 1 to 16 number of IMU from our custom control panel.

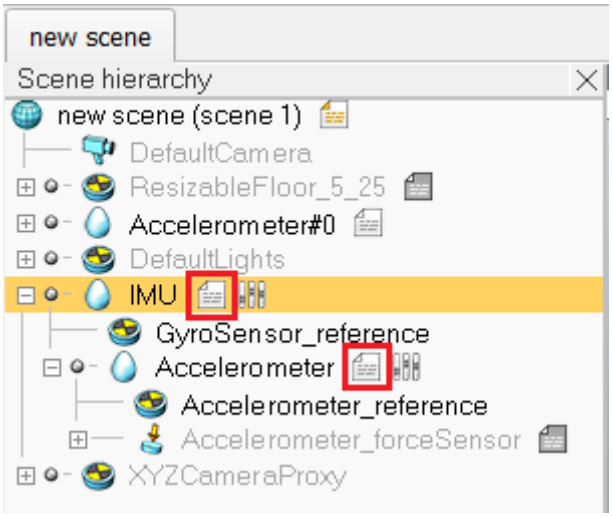


Figure 4.3: IMU Creation.

It is important to mention, the autogenerated script was in the non-threaded mode, if we write in nonthreaded mode all the data gets overwritten by the last sensor data. we hade to wrote threaded scripts get out of this situation.

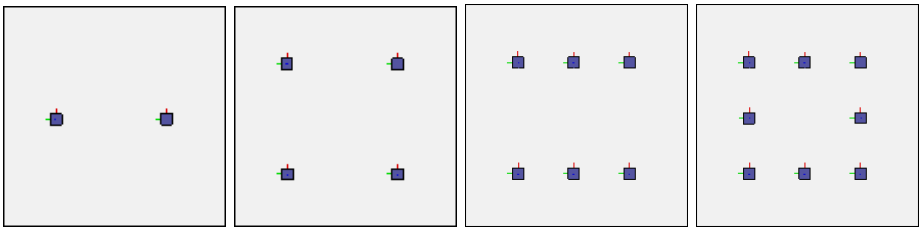
4.2.2 IMU pattern in a 2D plane

In sensor fusion, the number of IMU and the position of IMU plays a very important role. For that, it was very important for was to experiment with what number of IMU in which position gives us the best result. One of our primary objectives was to try the combination of IMU (2 to 16) in the simulation. It was very important to try different experiments with this number of IMUs for this particular issue scope. [..] have explored different avenues regarding 8 real IMUs joined on a circuit board to gather real-world information, yet not utilizing various examples with various numbers of IMUs. It is a direct result of the very complex nature of the task to connect and segregate IMUs after each and every run. At that point comes the issue of calibration; which is meticulous in

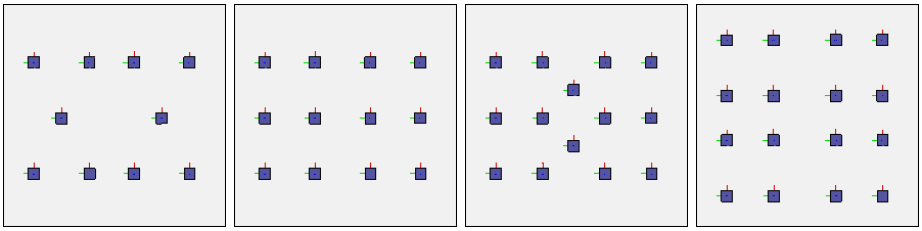
light of the fact that each time another IMU is joined or an IMU gets disengaged all the rest of the IMUs should be adjusted one by one.

Using a simulation instead of a real-world sensor comes handy in this type of situation. Just using a mouse click we can change the number of IMU from our custom control panel without needing any calibration or registration. To identify the patterns of IMU we need a bit of trial and testing since we didn't have any benchmark for IMU designs. We took []'s work with 8 IMUs as our base and characterized the examples for the IMUs around it. Moreover, we thought about the alignment and situating for various IMUs for a genuine gadget, which in the long run helped us to settle the examples.

The accompanying figures give a diagram of the pattern designs for the IMUs:



(a) 2 IMU pattern (b) 4 IMU pattern. (c) 6 IMU pattern. (d) 8 IMU pattern.



(e) 10 IMU pattern. (f) 12 IMU pattern. (g) 14 IMU pattern. (h) 16 IMU pattern

Figure 4.4: Number of IMU position on a 2D plane.

Although we initially try to test every pattern from 1 to 16 but we end up using an even number of IMUs for our purpose to reduce the time for CNN testing and there was no significant difference from changing 2 IMU to 3 IMU then 4 IMU.

4.3 Simulation Path

As we are prediction position and orientation of an object, we need a path in our simulation where our dummy object will go through. Therefore creating a simulated path as close as possible to real trajectory was one of the main concerns. Our path has to be in such a way where our object can move with 6Dof. In our simulation tool

the path controllers the position and orientation of the moving object in our case the 2D plane. We need to create the paths with multiple rotation and elevation where our object will move and we can collect the data of the position and orientation, in short, the pose of the object.

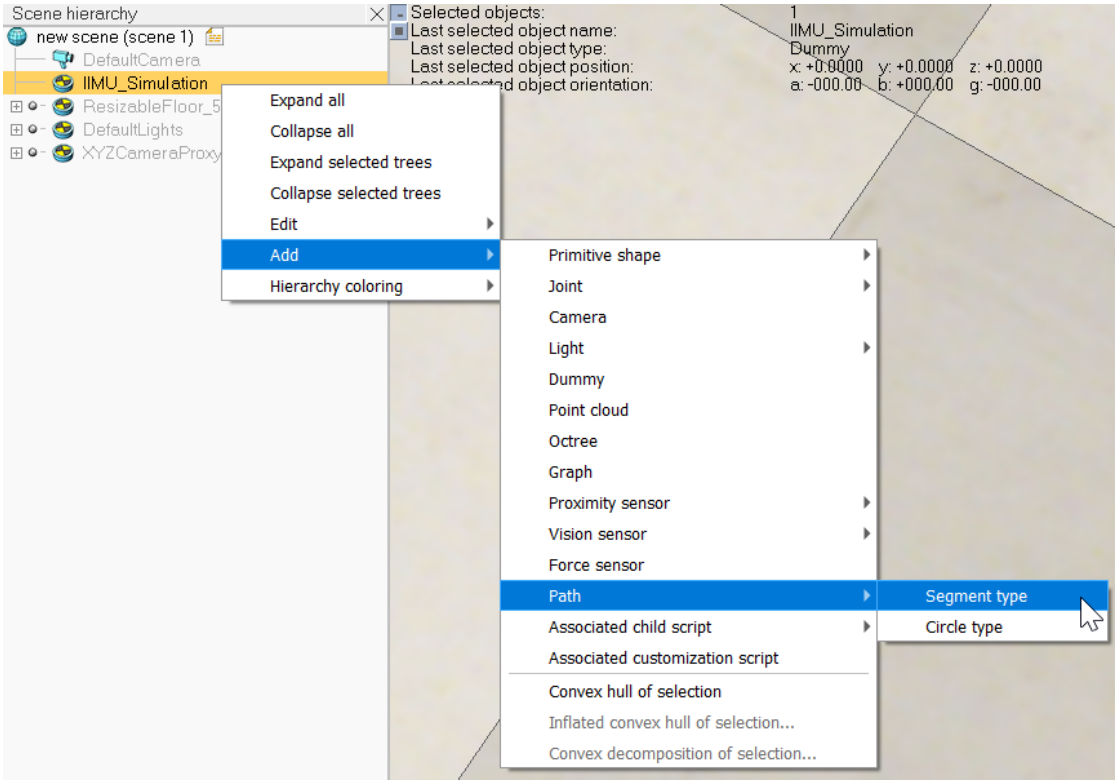


Figure 4.5: Path Creation.

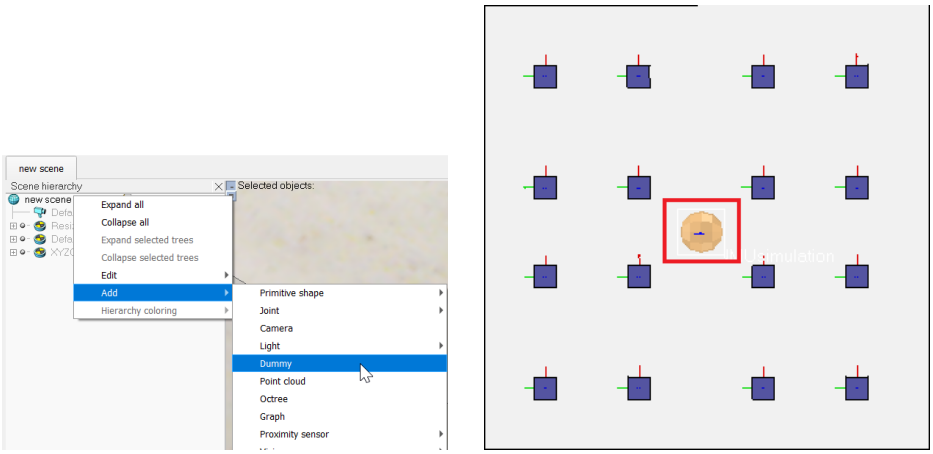
We need a large amount of dataset to train our Neural Network for that our path has to be long enough with orientation, with elevation gain and loss to generate the data. Meaning our plane with the IMU and dummy object will go up and down, left and right rotation along the path. These features need to ensure by the path design itself. The path design also facilitates other features slike roll, pitch, and yaw.

4.3.1 Dummy Object

To control our simulation environment we need a control panel, to do that we need an entry point. The creation of a dummy object makes the process easier. we need to add our customize script to the dummy object script to make our control panel. that's

why adding a dummy object is very important. The detail about the control panel will be discussed in the upcoming section. We can simply add a dummy object from the simulation tool using the following steps:

1. Navigate to the 'Scene hierarchy' pane.
2. Right click on the scene name.
3. Go to 'Add' and click on 'Dummy'



(a) Adding Dummy Object (b) Dummy Object in a Plane.

Figure 4.6: Dummy Object.

After creating the dummy object we named it IMU Simulation and we placed it in the center (x, y, z : 0, 0, 0) of the plane.

4.3.2 Creating the path for the plane

The path creation process was really time-consuming. In the beginning, we started just adding a default path from the simulation tool. There are 2 types of paths in the tool one is circular path another is the segmented path. As we needed path with many orientation and elevation we selected the segmented path showed in figure 4.5. The following steps are taken to add a segmented path in the scene:

1. Go to the 'Scene hierarchy' pane.
2. Right-click on IMU Simulation.

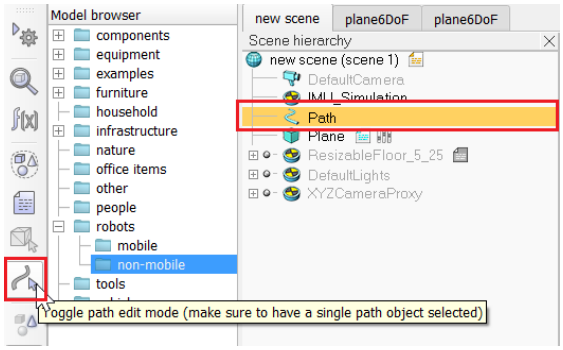
3. Go to 'Add' and then go to 'Path'.

4. Click on 'Segment type' to create the primitive path.

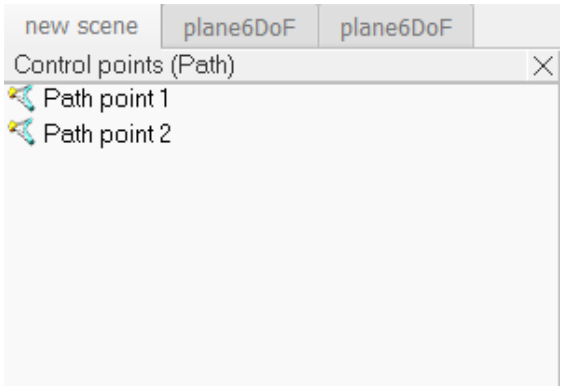
The segmented path just adds a straight line in the scene. At first, we added many segmented paths in the scene but when we tried to join them together it was cumbersome. Then we decided to modify a single segmented path by adding many path control points and give it random elevation, orientation, and trajectory. To do that we need to go to the section called Path edit mode.

4.3.2.1 Path Control Points

In order to manipulate our segmented path we took a single segmented path and created many path points. Adding path points is fairly easy on the simulation tool. Select the path from the scene.



(a) Adding Path Control Points



(b) After Path Controll Points are added.

Figure 4.7: Path Control Points.

1. Click the path edit mode shown in the figure 4.7
2. Go to the control point section and right-click on it.
3. From the menu, select Insert new control point after selection.

We repeated these steps many times to make our path longer as we needed.

The new point gets made right on the last point, or essentially put the point we chose to include the new point. To move the recently included point, from the outset we have to choose the point and afterward switch 'Object/Item shift' button on the top control panel. After we selected a control point in the scene, we just drag and drop in our desired position. This choice permits moving any article in the scene along with all the three-axis. Two kinds of movement are permitted through this process:

1. Move an object along the X-axes and Y-axes by using the mouse; by simple drag and drop.
2. Right-click on IMU Simulation.
3. Press 'CTL' in the keyboard and moving the cursor up and down to move the object along Z-axis.

We had to do these steps for every control points over our whole path.

4.3.2.2 Path Orientation

After we created the long path we run the simulation. The simulation was running well and we collected 20 minutes of simulation data. When we were analyzing the data we found that the object orientating in one axis. As we need 6Dof we can not take the data from the simulation.

After investigating we found to change the orientation of our plane we need to change in the path control points. But, the method or the ability to add a configuration script to adjust the path Control points orientation is not available automatically from the tool. We need to change every control point we added in our simulation path. This was a time-consuming process. The longer the path the more control point we had to change. After a considerable amount of time, we change the control points and run the simulation again. This time our plane was oriented along the 3 axes but there was a problem. The object was only changing its orientation only at the control point but not along the path. For better explanation lets say, The plane was moving from control point A to control point B then to control point C. The plane is only changing its orientation axis when it reaches the control point B and keeps its orientation along

the path From B to C and then change its orientation again in point C. This is not what we wanted.

We tried to solve this problem by turning on the "Automatic Orientation" checkbox in the path edit mode dialog box for orientation. By default, this box was checked. With this activated option, the object that moves along the path should frequently change its angles. But in simulation, it does not. The object only keeps the roll but not yaw and pitch. The characteristics of 6 DoF are thus violated.

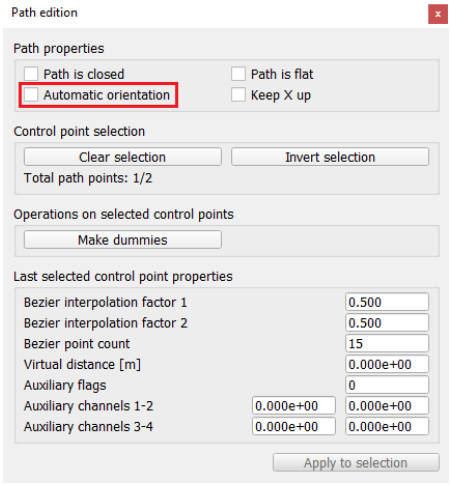
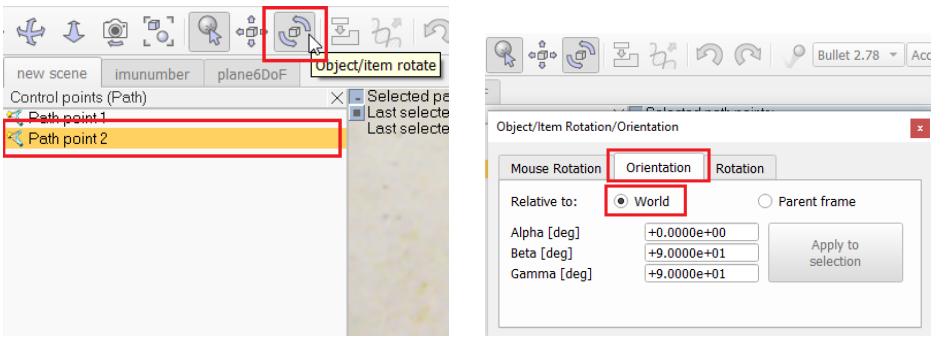


Figure 4.8: Toggle Autometic Rotation.

After a lot of investigation, the only solution we came up with is to add a lot more control points with a variant of angle values. To do that first we need to check out the 'Automatic Orientation' option from the Path edit panel. This enables us to move the object in our case the 2D pane with 6DoF. Creating control points with the very near proximity of multiple angles frequently, resembles the actual close to a real-world scenario. One thing that should be remembered here is that this configuration doesn't remove our missing Pitch, Yaw problem.

The following steps are necessary to set the orientation values by changing the control points' default angle values:

1. Select the latest point on the 'Control points' pane.
2. . Click the 'Object/Item' rotate on the top control pane. Doing this opens a new control panel for orientation and rotation for that particular control point.
3. Navigate to the 'Orientation' tab.



(a) Rotating the control point (b) Changing the Angles .

Figure 4.9: Orientation of control points.

4. Make sure orientation is relative to the world frame.

5. . Change the angle values (alpha, beta, and gamma) and click 'Apply to selection'.

After we fix all the contorl panel our path look like the followinge figure.

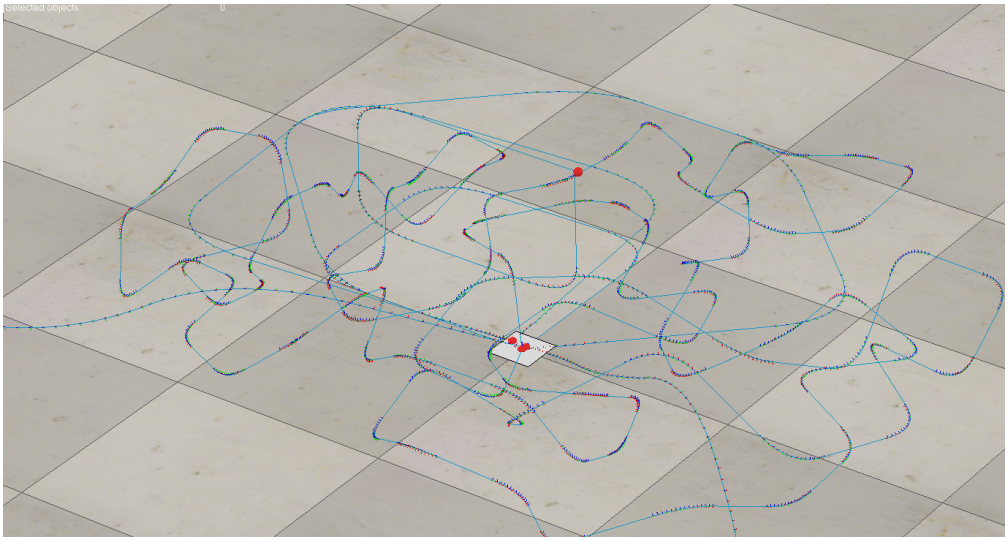


Figure 4.10: Final simulation path.

In the next section we will describe how we created a customize control panel for the simulation.

4.4 Customized Control Panel

We want to have a control panel to run our simulation. We could run to simulation from the 'CoppeliaSim' tool itself but we wanted to have some graphical user interface (GUI) to control our simulation. The customize GUI allows us to easily change the simulation environment such as Number of IMU, Object Velocity, Object Acceleration, and the Path should follow. Otherwise, we had to change the simulation script every time we run the simulation.

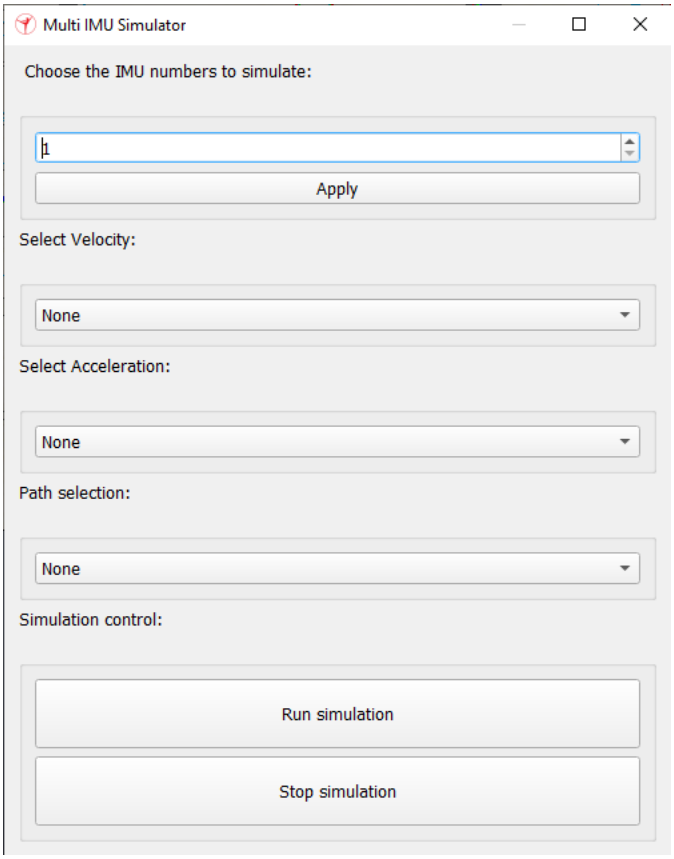


Figure 4.11: Simulation control panel.

Our control panel GUI was fairly simple and user friendly. We highlighted the 'Run simulation' and 'Stop simulation' button bigger to access easily because these are the buttons we used most. The other options were:

1. Number of IMU: this is a drop-down menu where we can easily select how many numbers of IMUs should be on the plane. The range is from 1 to 16 with

predefined positions.

2. Select Velocity: We had three predefined velocity option in this drop-down menu. The predefined options are 0.01, 0.03, 0.05 meter per second.

3. Select Acceleration: As like the 'Select velocity' menu we predefined the acceleration parameter in the dropdown menu.

4. Path selection: As we tested our simulation in multiple paths. In this dropdown section, we put the name of the path to select to run the simulation in that path. From the beginning, we created 6 different paths so we named them accordingly. Such as Path1, Path2 till Path6

4.4.1 Creating the options for the control panel

The GUI of the control panel created using simple XML. All the options from the GUI to work we need to add customized Lua Script in our simulation tool. As the starting point of our simulation is our Dummy Object, we had to add the customize Lua Script in the script of the Dummy object.

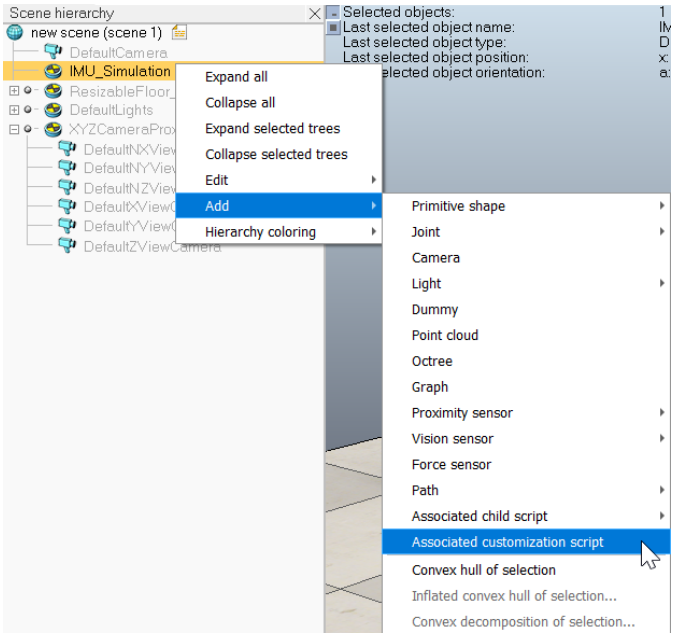


Figure 4.12: Customizing the script to create the control panel.

We added the necessary methods such as the functions for velocity and acceleration in this associated child script of the Dummy Object.

4.5 Data Collection

Collection of the Data for training our Neural Network model consists of two parts. One is collecting the data of the sensors(IMU) which will be our input data for the model and the other is collecting the data from the dummy object which we call the ground truth will be out label data for our model. Primarily we collected data both for ground truth and the IMU data out of the simulation as a txt file. The formate of the data will be discussed in the section: "" . Then we Fromate the data as per our requirement for the Neural Network Model.

It is important to note that, We spent a lot of time preprocessing the data to feed our model. The primary data set will be changed into as a csv formate after we preprocess the data as we need csv formate to feed our model in google collab []. We will discuss preprocessing in the Experiment setup [] section. It is very important to know about the shape of our data to know and identify how our data is going through the network. That is why we took every sensor data in different file formate. Without knowing the shape and size of the data without it, data processing and feeding into the neural networks are incredibly difficult.

4.5.1 Sensor (IMU) Data

We collected each of the Sensor data in different .txt file. The number of the data file, in this case, depending on the number of IMU we have in our simulation. For example, if we have 4 IMU in our simulation we have 8 data files. 2 files for each IMU, one is for the gyroscope data and the other for the accelerometer data. We could have collected the data in a single file but we decided to separate the initial data because it will help us to preprocess the data for better reliability and manipulation. The data formate for the sensor data is given below.

Data formate for the accelarometer :

timestamp_1	a_x	a_y	a_z
timestamp_2	a_x	a_y	a_z
....	....	....	....
....	....	....	....
timestamp_n	a_x	a_y	a_z

Data formate for the gyroscope:

The explanation for calling columns a_x, a_y, and a_z of the accelerometer is that 'a' is the unit of acceleration and is readable easily. The same applies to the columns of gyroscopic data "ω" as the angular velocity unit.

timestamp_1	ω _x	ω_y	ω_z
timestamp_2	ω _x	ω_y	ω_z
....	....	....	....
....	....	....	....
timestamp_n	ω _x	ω_y	ω_z

4.5.2 Ground Truth

As we said before the ground truth data is the label for the training of our model. The word 'ground truth' in machine learning refers to the precision of the evaluation of learning methods by the training sets. It is used to support or deny scientific theories in mathematical models. In terms of our goal, the ground truth is the precise position and orientation (pose) of the object in a particular timestamp. We took milliseconds difference from each timestamp.

In our training data set, the input data will be feed into the model and try to mimic the result as close as possible with the ground truth. Ground truth values are extremely crucial to the validation process. So we have to be very careful to have the precise ground truth values from the dummy object in our simulation. Another important point to note, The number of ground truth readings should be equal to the number of IMU reading, this way we tried to somewhat synchronize the IMU and ground-truth data. The explanation behind this is that "NaN" values should not be obtained by using a test path dataset to determine the efficiency of one of our trained models.

Normally we should get the ground truth data from the script written in the dummy object attached to the plane but we faced problems with the ground truth values of the script attached to the plane. For unexplained reasons, several values were absent. The CPU was not able to sense the IMU sub-script, which is the accelerometer script, as we thought. We carried out further experiments to integrate the ground truth script into the IMU. After several tests, the ground truth value was finally obtained by using the script in the associated Gyroscope script with similar data point counts. We used the gyroscope script as the parent IMU script.

Data formate for the Ground Truth:

timestamp_1	t_x	t_y	t_z	o_x	o_y	o_z
timestamp_2	t_x	t_y	t_z	o_x	o_y	o_z
....	....	....	....	....	....	....
....	....	....	....	....	....	....
timestamp_n	t_x	t_y	t_z	o_x	o_y	o_z

We have 1 data file for the ground truth for each simulation run. The file contains

seven data columns. The 1st one is the timestamp in milliseconds, the next three are the position coordinates of the plane across three axes and the last three columns contain the plane's orientation across three axes in degrees. Seven data points in each row make the pose of the plane. It is important to note the orientation calculated in relation to the "World frame" which is the whole simulation frame.

5 Experiment setup and Methodology

In this chapter we will discuss about our experiment setup and methodology for training our neural network models. Our goal is to preprocess the data from our simulation software in a format that we can feed it to the neural network models, then train the models. The data we got from our simulation is not suitable for direct feeding to the neural network because different models have different structure for the input of the data. We will also discuss the mythology behind the formatting of the data and our experiment setup and also the architectures of our Convolutional Neural Network (CNN).

5.1 Data Preprocessing

Our data preprocessing mainly contain 2 steps. First align the IMU data points (number of rows) according to the data points of our Ground Truth. Then fine tune the data in order to make it suitable for calculation of the CNN.

5.1.1 Aligning the data rows

As mention before in the section 4.5 we took our IMU data and Ground Truth data in different files. For each IMU we have 2 files for accelerometer data from the accelerometer sensor and angular velocity data from the gyroscope. The Ground Truth data has one file for each simulation run. The first thing is to check is the now of rows we got from each file. Our goal is to ensure that the count of data rows in the acceleration and angular velocity files was identical to the count of data rows in the GT file to ensure that they were fed smoothly into the model and keep the data sync with timestamps. Also, we wanted to ensure there are no “NaN” values in the data set to ensure that our model could learn optimally.

Our target was to get 100 thousand data rows. For that, we needed to run the simulation for more than an hour. We were expecting to get the same number of data rows from each file but unfortunately, that was not the case. We have a similar number of data rows form each IMU accelerometer and gyroscope, but the GT data rows were a bit less than the number of data rows from the IMU. For example, if we get ‘n’ number

of data rows from each IMU file we were getting 'n-x' number of data rows from the GT.

Our assumption behind this problem is that when we stop the simulation by clicking the stop button The whole simulation doesn't stop immediately. The plane stop moving but the Sensor still keeps generation some data on the other hand the GT stop generation data the moment we click stop because the GT script is a nested script, unlike the IMU script. To overcome this problem, we used a simple solution. We simply removed x data points from the data files of the accelerometer and the gyroscope to make all data files compatible with the same data point count. We measured the value of x is approximately 0.04% of all data points which is ignorable.

5.1.2 Data Tuning

The next task is to tune the data in way that its suitable for feeding into the neural network model. Our simulation generated data is a numerical value consists variable number of digit after decimal point. We tried to feed the data into neural networks, but it prevented us from achieving optimal accuracy with many digits after the decimal point. The predicted values are not counted as a correctly forecast value because of small change in a given digit. We rounded up each data values to 4 digit after decimal point which gives us a small advantage in the accuracy of prediction and evaluation of the overall model.

	tx	ty	tz	ox	oy	oz
Row n	-0.187795669	1.460141063	0.024999999	0.006407991	1.227231145	-0.063147597
Row n+1	-0.189894766	1.461498737	0.024999999	0.023488231	1.222845793	-0.069066621

Table 5.1: Sample ground truth data before rounding up

Table 5.1 and Table 5.2 show the effect of change when the values are rounded. The new data looks better and we were able to test our models more effectively.

	tx	**ty**	**tz**	**ox**	**oy**	**oz**
Row n	-0.1877	1.4602	0.0250	0.0065	1.2273	-0.0631
Row n+1	-0.1898	1.4615	0.0250	0.0235	1.2229	-0.0690

Table 5.2: Sample ground truth data after rounding up

5.2 Methodology for processing data for the input

As our ground truth data values provide us with the pure pose (position and orientation) of the object. To train our model we wanted to predict the difference of pose for a particular timestamp using the previous timestamp data from the sensors (IMUs). For example, for time t_0 we have the object pose pt_0. In time t_1 we will have the object pose pt_1 so our pose change from time t_0 to t_1 (Δ t) will be Δ p. We want to predict this delta p using the IMU data at time t_0 and t_1. It became obvious when we see the equation for velocity, acceleration, and angular velocity.

$$vt = vt_0 + a * \Delta t \tag{5.1}$$

equation 5.1 is the standard equation for the velocity of an object in time t. where vt_0 is the velocity in time t_0 and a is the acceleration and delta t = t_1 -t_0 .

$$a = \frac{v_t - vt_0}{\Delta t} \tag{5.2}$$

from equation 5.2 we can see that acceleration depends on the time delta and the velocity delta.

$$p_t = p_{t0} + v_t \tag{5.3}$$

Equation 5.3 is the standard equation for the calculation position at time t.

$$s = v * t \tag{5.4}$$

$$\Omega = \frac{v}{r} \tag{5.5}$$

Equation 5.4 , which also depends on velocity, is the standard equation for distance. Equation 5.5 is the normal angular-speed expression, which depends also on the distance. R in this equation, the radius is indicated if the circle created at that time. It gave us the concept of using time differences (always) between two data points rather than the actual time. Using the following basic equation, we replace the raw time (in milliseconds) of our data using time delta.

$$\Delta t = t_{i+1} - t_1 \tag{5.6}$$

We have decided to use the position delta as our labels because we calculate the time frame differences. This applies to the position and orientation of the object, the plane in our case. This helps us to assess how much displacement in a given time interval has been made in the position and orientation of the plane. The fact is that, instead

of the actual position and orientation values in time, it makes more sense to calculate the difference. Since orientation and position may have similar values at different timestamps.We used Equation 5.6 to get the time difference.

$$\Delta p = p_{i+1} - t_1 \tag{5.7}$$

$$\Delta o = o_{i+1} - o_1 \tag{5.8}$$

Equation 5.7 and 5.8 represent the method of measuring position and orientation displacements. By following this method, we feed displacements as labels to the neural networks. The network will predict the displacements after the training is complete. The Euclidian distance can also be calculated.

5.3 Convolutional Neural Network Architecture

We use various combinations of architecture to evaluate our solution. We created the combinations based on the different hidden layers, number of neurons, number of channels in the input layer. We obviously played with the different hyperparameter for each architecture to make its performance better but we did not see as a major architectural difference. We will describe the hyperparameter at the index <()>.

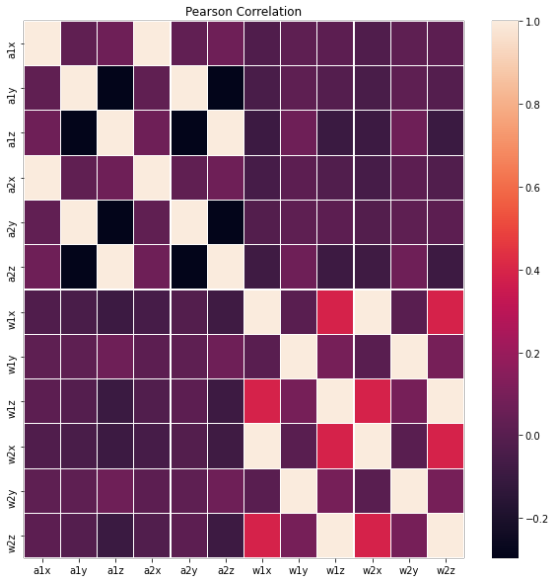
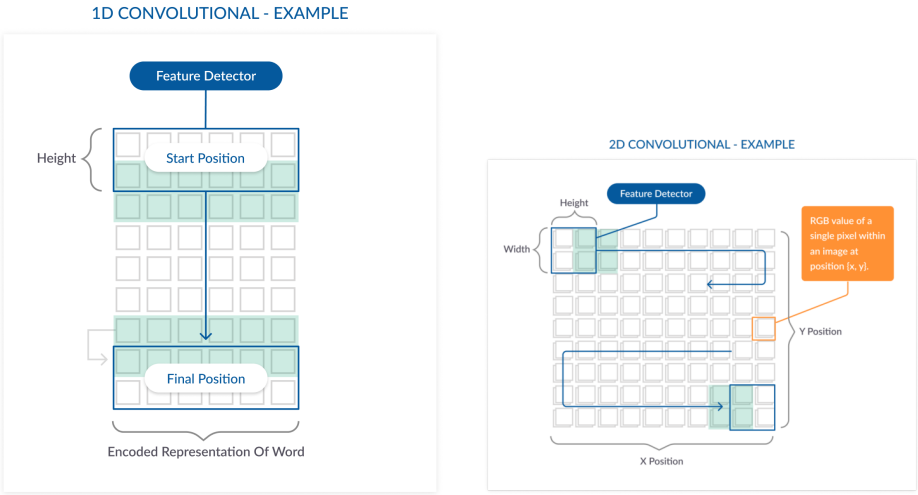


Figure 5.1: Pearson correlation of IMU-2 dataset.

Before we discuss our CNN architecture we wanted to check if our dataset is suitable for regression analysis as we do not have the problem of multiple variables depending on each other or in other words, multi-colinearity. For that, we did Pearson correlation to visualize the correlation among the variables in our dataset. From the figure 5.1 we can see that our data does not have that problem.

Our basic CNN architecture starts with a (w * h) input layer then 'n' hidden CNN layer and the output is always a dense layer with 6 neurons as we need the output of 6 data points. It is important to note that, instead of the 2D convolution layer we used the 1D convolution layer because the 2D convolution works very well for image and video data where 1D convolution is best suitable for time series data. The main difference between 1D and 2D convolution is that in 1D convolution the kernel slides along one dimension wherein 2D convolution the kernel slides along 2 dimensions (width and height)

The figure **??**fig:diff1D2D) shows the difference between 1d and 2d convolution

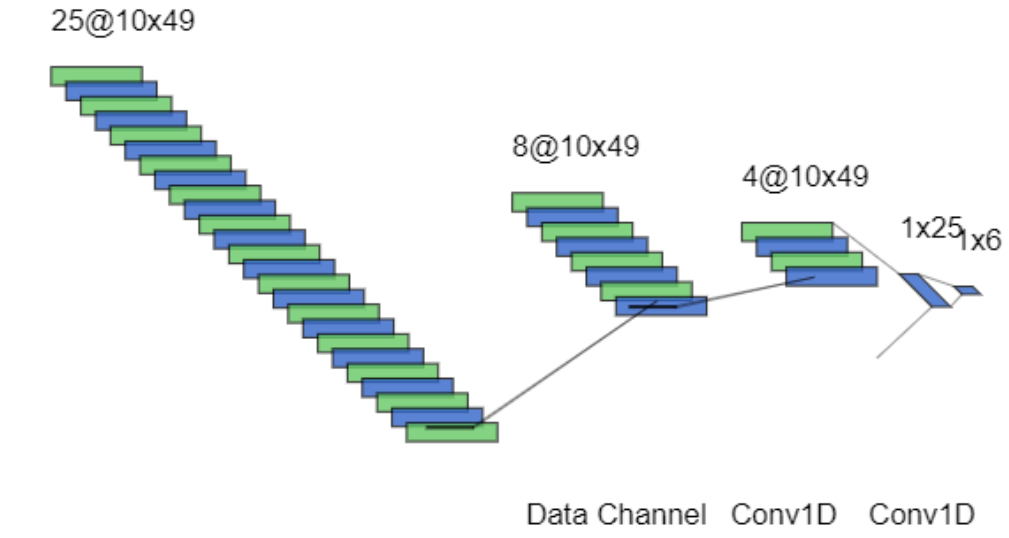


(a) 1D convolution architecture (b) 2D convolution architecture

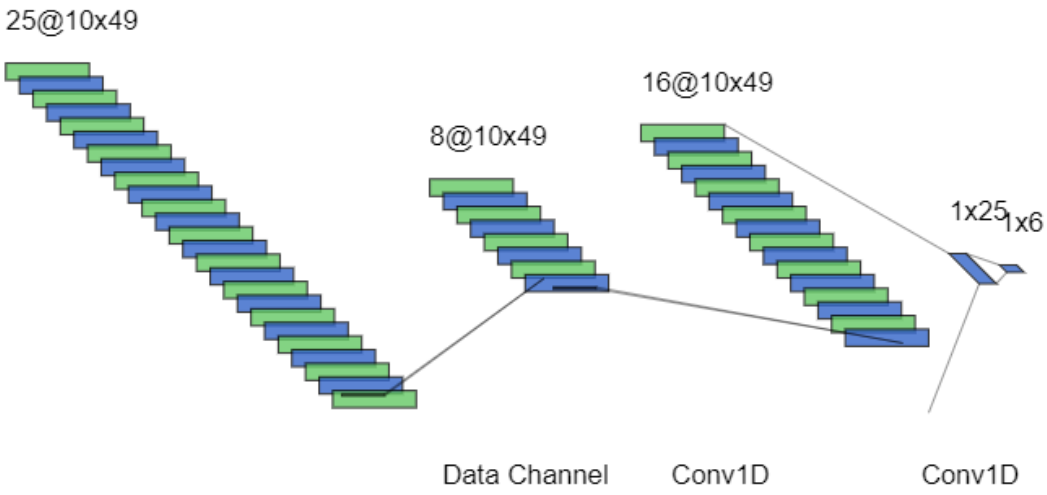
Figure 5.2: Difference between 1D and 2D convolution.

We can divide our CNN architecture into two major categories Expanding model and Shirking model. As the name suggests the expanding model increases the number of neurons for each layer and the shirking model decreases the number of neurons for each layer. For each shrinking and expanding model we also experiment with different numbers of hidden layers. For example, the shrinking model we had 2 convolution layers and 3 convolution layers the same with the expanding model. The figure shows the visual representation of the shrinking and expanding models of 2 convolutional

layer.



(a) Shrinking model architecture



(b) Expanding model architecture

Figure 5.3: Visual representation of shrinking expanding model architecture.

We did not use dropout in our architecture mainly because dropout usually used to prevent overfitting and it turns off some neuron to do that. Because of that, some data might not get calculated in the CNN. In our situation, we are not finding features from our data so that we can not prioritize some sections and drop some sections of data in the calculation. Instead of dropout, we used batch normalization to overcome the problem of overfitting

5.4 CNN Parameters

will be discueed

5.5 Loss Function

A typical neural network is developed by stochastic gradient optimization algorithms and weights modified by algorithm back propagation. The word 'gradient' is a gradient of mistake. The loss function or target function is used to evaluate the solutions for the candidate during optimisation. The goal is to minimize the error and the function is called a loss or cost function.Since our problem is a regression problem, there are two viable loss functions to optimize our algorithms. Mean Absolute Error (MAE) and Mean Squared Error (MSE).

Yi is the real label value of these equations, and (y i) da is the value expected. Such two failure features have their own benefits and disadvantages. For MAE, if there are more outliers it works better because there are no square operations. Average all distances. On the other side, since there is a square procedure MSE penalizes long distances. For broad values, MSE has a higher gradient. MAE contribute to cost optimisation and therefore to the network's constant learning speed. For MSE, that's quite the opposite. Our dataset includes not so many outliers, so MSE performed on our unique dataset more effectively. This doesn't mean MAE can't be done at all. This doesn't mean MAE can't be done at all. For MSE we performed in terms of training time and predictive efficiency a little better.

5.6 Optimizer

One of the arguments needed to compile a model in Keras is an optimizer. An significant parameter since optimizers determine how the cost function of neural networks is the. The goal is to find the minimum. For neural networks there are many optimizers available, such as –Stochastic Gradient Descent (SGD), Root Mean Square

Propagation (RMSProp), Adaptive Moment Estimation (Adam), Nesterov-accelerated Adaptive Moment Estimation (Nadam)) and others. The goal is to find the global minimum at which the learning cost is the lowest. Although the gradient decent attempts to optimize the cost or loss function without updating the weight deviation, it could hover over the optimal values instead. Via its capacity to change the weight at every data point, the solution is therefore to use SGD. SGD is problem-sensitive too, however, since the batch size is too big to predict right. SGD's Mini-batch SGD and SGD had implemented a momentum in order to overcome this problem. There is even an acceleration parameter which supports SGD in its optimization.Yet weight-update gets very fast when dynamic and speed-up are mixed, confounding and anticipating the network. A comparatively better solution is Adagrad, which is able to update weights for every parameter in a very good direction. The learning rate parameter will then decrease to a point in which the parameters are no longer changed and the learning process will end.

Adam for deep learning is one of the most commonly used and effective optimizers. This is close to the method by which the optimizer adds momentum. The learning process seems slow at the beginning, but it gets faster after some time. For various parameters to change Adam may also take a different time step. It leads to faster convergence with the help of momentum. Since Adam is a common optimizer, we decided to adjust the learning rate for Adam over SGD with the default adam parameters. We agreed in several iterations to use a chart of leaning rates for a single standard. We chose to use a list of leaning rate for a single model in multiple iterations.

lr = [0.01, 0.001, 0.0001]

Initially all the other parameters of adam such as – beta1, beta2, epsilon, decay were remained with the default values.

5.7 Data structure for CNN

We decided to use a number of different data channel for the input layer, Our main motivation was to determine how many rows of sensor data is needed to predict accurately the pose. Our initial approach was to use every row of input data (IMU) to predict every row of pose data (GT). (table here) But CNN doesn't work that way. CNN models try to convolute the data and try to find the best feater in the data set for that purpose 1 by 1 approach don't work. Then we decided to divide the data into channels. For example, we used n number of data rows to predict 1 pose. Here n could be 2, 5, 10, and 20, if we used 5 rows of IMU data as an input channel then we tried to predict the 6th row of pose (data). As in for every 5n data rows for the input we try to predict (5n+1)th row of pose data The image below shows the data channel. We

used this method for 10n, 15n and 20n to determine the optimal data channel It's very important to note that we used this architecture to train our model for 2, 4, 8 and16 IMU.

6 Experiment Results and Comparison

In this chapter we analyze the tests, the possible outcomes and the best network configurations for the tests of our experiments. The IMU-2/4/8/16 data sets have been used for our model testing. We think that training with our data sets is enough to allow a choice for a particular system , for example by using particular IMU numbers instead of 2/4/8/16. Of those four datasets, we have trained 128 models and validated the models with invisible test results. The evaluation of these models gave us a good understanding of the efficiency and the usefulness of the model in predicting the object's location.

For the assessment, we decided to concentrate on preparation and loss of validity, which uses the loss functions of MAE and MSE. The prediction accuracy of models is also tracked and measured. The calculation of prediction accuracy makes little sense because so many floating points have to be taken into account when calculating accuracy. The prediction precision (both position and orientation) of displacements gives us a decent value which we can rely upon because of its many floating points. Thus we decided to evaluate our models by measuring deviations from the displacement predictions.

This allows us to easily understand the performance of the model. In addition, after predicting the Euclidian deviation distance of the results can also be calculated. We used boxplots to consider the distribution to represent the error deviations of the predictions. We also agreed to take the outliers away, so we didn't have too many.

Following the model evaluation, we intended to combine our solutions in order to have a higher level overview with the best and worst models. In the following sections we will analyze the results of testing our trained models, broken down by network form and data set (depending on number of IMUs used. The data sets are used to train various network types, e.g., shrinking and expanding, 2 and 3 levels of CNN layey architecture, specific learning rates and optimizer parameters.

6.1 (

Results for the Shrinking models)

6.1.1 (

Result with multiple channel)

6.1.2 (

Result with multiple CNN Layer)

6.2 (

Results for the Expanding models)

6.2.1 (

Result with multiple channel)

6.2.2 (

Result with multiple CNN Layer)

List of Figures

List of Tables