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

Some Introduction will go here

1.1 Problem and solution overview

1.1.1 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[] shows that increasing the number of IMU tends to provide better accuracy.

1.1.2 Goals

Our goal is to develop a deep learning model primarily with Convolutional Nural Network (CNN) and Dense Network to fuse multiple IMU data in a simulated environment. We will use many combinations of several IMUs and different CNN networks to predict the position and orientation of an object in a simulated environment and compare the results with other Deep Learning technics such as Recurrent Neural Network (RNN). We will take the combination which provides the best accuracy in

the simulated environment and apply that model in a real-world scenario to test its accuracy to predict object pos.

Acceleration values from an IMU are well known to be highly unstable and there could be misalignment between axes. Using Deep Learning, we aim to achieve a stable and reliable position and orientation transformation without any camera, sensor calibration, registration, and prior error correction.

1.2 Structure of the Thesis

This document is divided in five chapters. In the second chapter, the necessary theoretical background is presented, whereas third chapter shows the structural solution and implementation. Chapter four introduces the obtained results and pertinent analysis, while the conclusions and proposed future work are summarized in chapter five. Afterwards, 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

A sensor is a device that can monitor the environment that its designed for. Senson can be a single pice of hardware or can be a combination of hardware accuaring multiple information. Sensors vary in quality and price. Many modern sensors have on-board processing units to understand the data acquired from the sensing element without a separate computational platform. Remote sensing devices are sensors that can perceive information without physical contact. Unfortunately, sensors tend to have inherent problems including, but not limited to [Elmenreich, 2001]:

- Spatial coverage limits: Each sensor may only cover a certain region of space. For example, a dashboard camera will observe less surrounding region than a camera with a wide-view lens.
- Temporal coverage limits: Each sensor may only provide updates over certain periods of time, which may cause uncertainty between updates.
- Imprecision: Each sensor has limits to its sensing element. • Uncertainty: Unlike imprecision, uncertainty varies with the object being observed rather than the device making the observation. Uncertainty may be introduced by many environment factors or sensor defects in addition to time delay.
- Deprivation of sensor: Sensor element breakdown will cause loss of perception in environment. Many different types of sensors exist in the world, and each has its own unique ap plication. Four sensors are pertinent to the field of autonomous driving [Levinson et al., 2011]. The four sensors, their descriptions, uses, advantages and disadvantages are men tioned below:
- GPS: Global-positioning system (GPS) is a system of satellites and receivers used for global navigation of Earth designed by the U.S. military. GPS sends a signal to any GPS receiver with an unobstructed line of sight to four or more GPS satellites surrounding Earth [gps, 2011]. GPS is useful for finding the exact coordinates of a vehicle when it is in the line of sight of multiple satellites orbiting the Earth.

– Advantages: Precise coordinate measurements, fast, reliable in line of sight, externally-managed satellite systems. – Disadvantages: Expensive, subject to failure in bad weather conditions, subject to failure in distant locations where satellite coverage is blocked or unavailable, dependent on external data source, subject to hijacking and interference.

- Radar: RAdio Detection And Ranging (radar) is a remote sensing device that uses an antenna to scatter radio signals across a region in the direction it is pointing and listens for response signals that are reflected by objects in that area. Radar measures signal time of flight to determine the distance. Radars may use the doppler effect to compute speed based on shift in frequency of scattered waves as an object moves. Radar is useful for detecting obstacles, vehicles and pedestrians around a vehicle [Huang et al., 2016]. Tracking multiple targets at once is a primary use for an automotive radar. – Advantages: High-bandwidth signals, wide-spread area coverage, inde pendent from external systems, works in multiple weather conditions, light independent solution. – Disadvantages: Expensive, subject to interference, easy to corrupt signal with electromagnetic interference, many reflective radio responses make it harder to manage radar signals, algorithms for radar tracking are still imperfect, narrow field-of-view.

- Camera: A camera is an optical instrument that utilizes at least one converging or convex lens and a shutter to limit light intake into an enclosed housing for capturing images or recording image sequences [Kodak, 2017]. Video cameras work much like still-image cameras, but instead of simply capturing still images, they record a series of successive still images rapidly at a specific frame rate

  [Kodak, 2017]. A camera is useful for acquiring images or video sequences of object pixels in view of the lens in order to help detect, segment, and classify objects based on perceivable object properties like location, color, shape, edges and corners. – Advantages: Perceives high-level object characteristics like color, shape, and edges, perceives location relative to camera unit, easy to visualize data. – Disadvantages: Potential slow frame-rate update, image quality may be dependent on light, weather and various other factors, data-intensive processing, all cameras perceive objects differently, typically has a limited range of perception compared to other sensors, may be expensive.

- LiDAR: Light Detection and Ranging (LiDAR) is a method of remote sensing that uses light in the form of a pulsed laser to measure distance to an object based on signal time of flight [NOAA, 2012]. LiDAR is useful for perceiving surroundings when 3-dimensional, high-resolution, light-independent images are necessary. – Advantages: Independent of light, weather and external data sources, fast,

accurate, 3-dimensional, high-resolution. – Disadvantages: Expensive, subject to interference by reflection or lack thereof, incompatible with transparent surfaces, data-intensive processing, less durable than other sensors.

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 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 algo-

rithm. 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

2.7 Sensors

A sensor is a piece of hardware that monitors an environment based on a sensing element. Sensors vary in quality and price. Many modern sensors have on-board processing units to understand the data acquired from the sensing element without a separate computational platform. Remote sensing devices are sensors that can perceive information without physical contact. Unfortunately, sensors tend to have inherent problems including, but not limited to [Elmenreich, 2001]: • Spatial coverage limits: Each sensor may only cover a certain region of space. For example, a dashboard camera will observe less surrounding region than a camera with a wide-view lens. • Temporal coverage limits: Each sensor may only provide updates over certain periods of time, which may cause uncertainty between updates. • Imprecision: Each sensor has limits to its sensing element. • Uncertainty: Unlike imprecision, uncertainty varies with the object being observed rather than the device making the observation. Uncertainty may be introduced by many environment factors or sensor defects in addition to time delay. • Deprivation of sensor: Sensor element breakdown will cause loss of perception in environment. Many different types of sensors exist in the world, and each has its own unique ap plication. Four sensors are pertinent to the field of autonomous driving [Levinson et al., 2011]. The four sensors, their descriptions, uses, advantages and disadvantages are men tioned below: • GPS: Global-positioning system (GPS) is a system of satellites and receivers used for global navigation of Earth designed by the U.S. military. GPS sends a signal to any GPS receiver with an unobstructed line of sight to four or more GPS satellites surrounding Earth [gps, 2011]. GPS is useful for finding the exact coordinates of a vehicle when it is in the line of sight of multiple satellites orbiting the Earth.

– Advantages: Precise coordinate measurements, fast, reliable in line of sight, externally-managed satellite systems. – Disadvantages: Expensive, subject to failure in bad weather conditions, subject to failure in distant locations where satellite

coverage is blocked or unavailable, dependent on external data source, subject to hijacking and interference. • Radar: RAdio Detection And Ranging (radar) is a remote sensing device that uses an antenna to scatter radio signals across a region in the direction it is pointing and listens for response signals that are reflected by objects in that area. Radar measures signal time of flight to determine the distance. Radars may use the doppler effect to compute speed based on shift in frequency of scattered waves as an object moves. Radar is useful for detecting obstacles, vehicles and pedestrians around a vehicle [Huang et al., 2016]. Tracking multiple targets at once is a primary use for an automotive radar. – Advantages: High-bandwidth signals, wide-spread area coverage, inde pendent from external systems, works in multiple weather conditions, light independent solution. – Disadvantages: Expensive, subject to interference, easy to corrupt signal with electromagnetic interference, many reflective radio responses make it harder to manage radar signals, algorithms for radar tracking are still imperfect, narrow field-of-view. • Camera: A camera is an optical instrument that utilizes at least one converging or convex lens and a shutter to limit light intake into an enclosed housing for capturing images or recording image sequences [Kodak, 2017]. Video cameras work much like still-image cameras, but instead of simply capturing still images, they record a series of successive still images rapidly at a specific frame rate

[Kodak, 2017]. A camera is useful for acquiring images or video sequences of object pixels in view of the lens in order to help detect, segment, and classify objects based on perceivable object properties like location, color, shape, edges and corners. – Advantages: Perceives high-level object characteristics like color, shape, and edges, perceives location relative to camera unit, easy to visualize data. – Disadvantages: Potential slow frame-rate update, image quality may be dependent on light, weather and various other factors, data-intensive processing, all cameras perceive objects differently, typically has a limited range of perception compared to other sensors, may be expensive. • LiDAR: Light Detection and Ranging (LiDAR) is a method of remote sensing that uses light in the form of a pulsed laser to measure distance to an object based on signal time of flight [NOAA, 2012]. LiDAR is useful for perceiving surroundings when 3-dimensional, high-resolution, light-independent images are necessary. – Advantages: Independent of light, weather and external data sources, fast, accurate, 3-dimensional, high-resolution. – Disadvantages: Expensive, subject to interference by reflection or lack thereof, incompatible with transparent surfaces, data-intensive processing, less durable than other sensors.

2.8 Inertial Mesurement Unit (IMU)

An inertial measurement unit is an electronic device that measures and reports a body's specific force, angular rate, and sometimes the orientation of the body, using a combination of accelerometers, gyroscopes, and sometimes magnetometers.

2.9 Sensorfutions

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 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 mea surements 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.

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

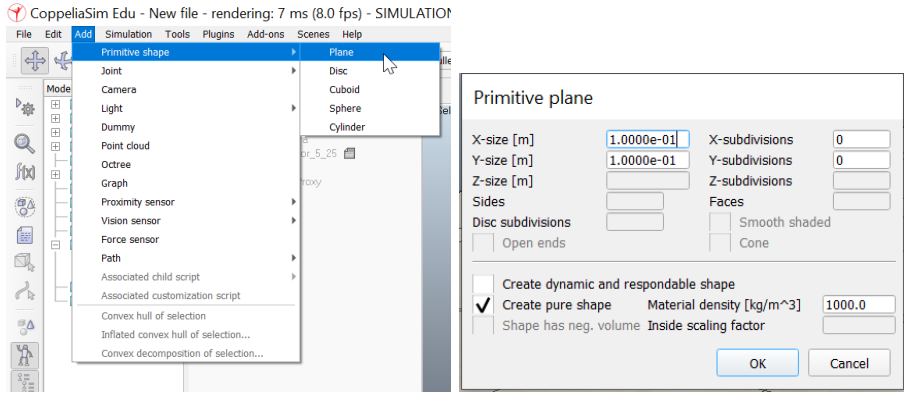
To recreate a real-world like situation, the main thing we required was an object for the scene. The object would represent the qualities of a moving segment in the simulated scenario

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.

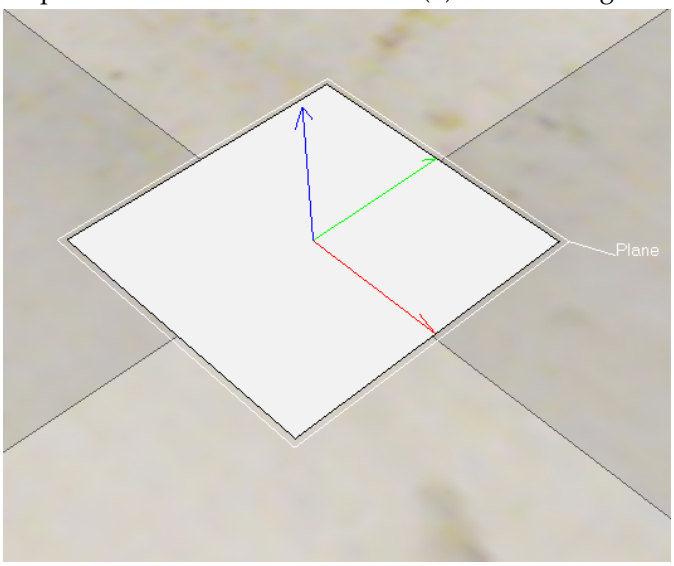
The plane is sufficiently large to contain at least 16 IMUs which is the most noteworthy number of IMUs we chose to utilize. The examples for putting the IMUs on the plane had been chosen together which would in the end help us to get the most ideal and solid information.

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'.

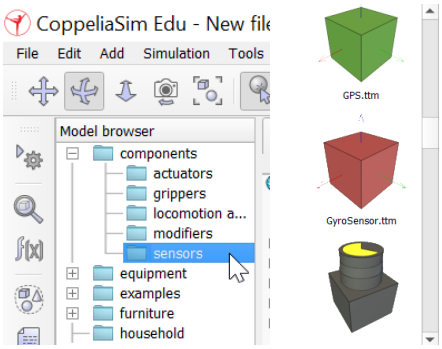
4.1.3 Inertial Measurement Units

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.

4.1.3.1 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.

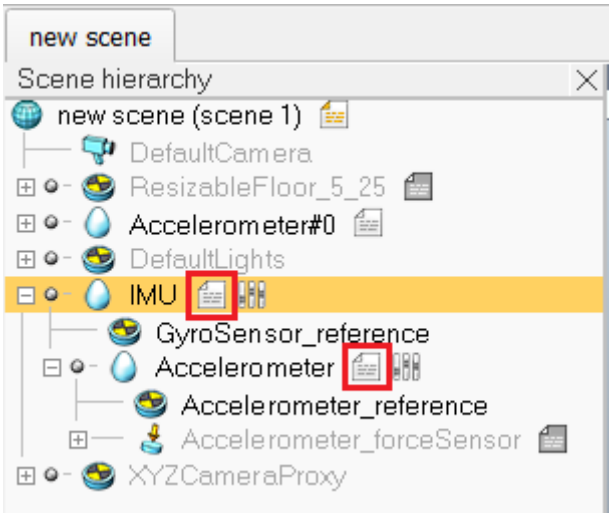


Figure 4.3: IMU Creation.

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.

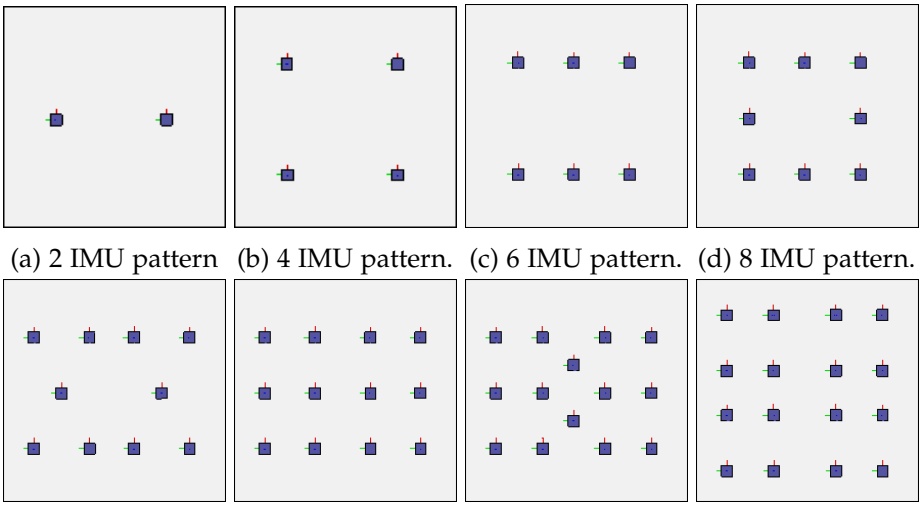
4.1.3.2 IMU pattern in a 2D plane

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.

4.1.4 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.

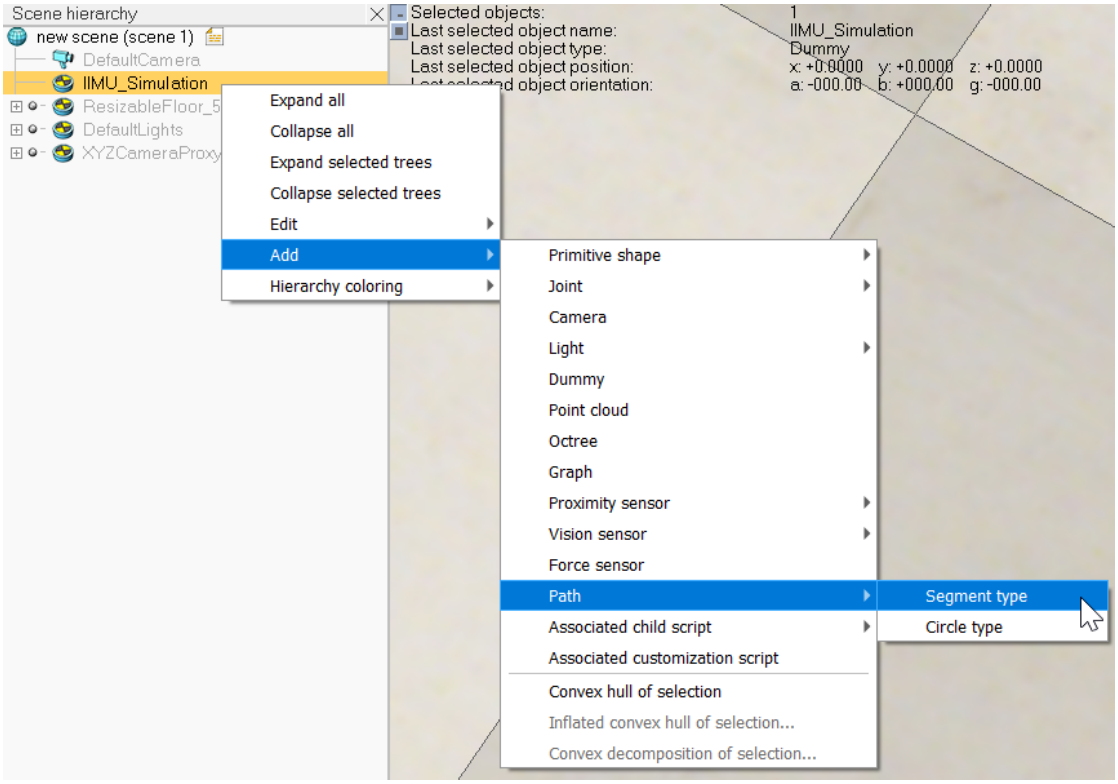


Figure 4.5: Path Creation.

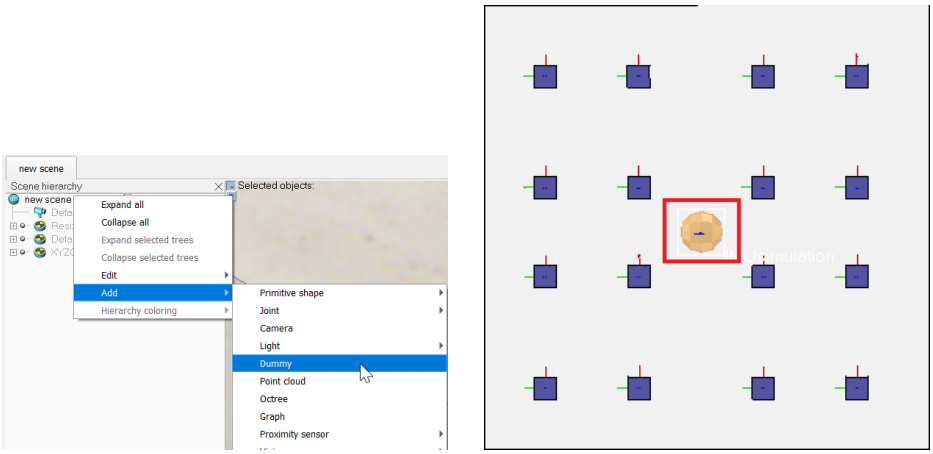
We need a large amount of dataset to train our Neural Network for that our path has to be long enough with orientation, with elevation gain and loss to generate the data. Meaning our plane with the IMU and dummy object will go up and down, left and right rotation along the path. These features need to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

4.1.4.1 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.

4.1.4.2 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.

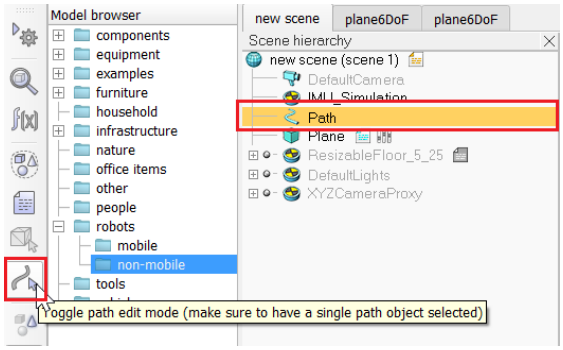
3. Go to 'Add' and then go to 'Path'.

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

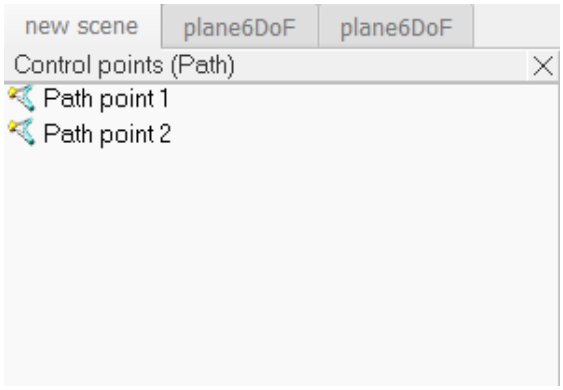
The segmented path just adds a straight line in the scene. At first, we added many segmented paths in the scene but when we tried to join them together it was cumbersome. Then we decided to modify a single segmented path by adding many path control points and give it random elevation, orientation, and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

4.1.4.3 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.

4.1.4.4 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.

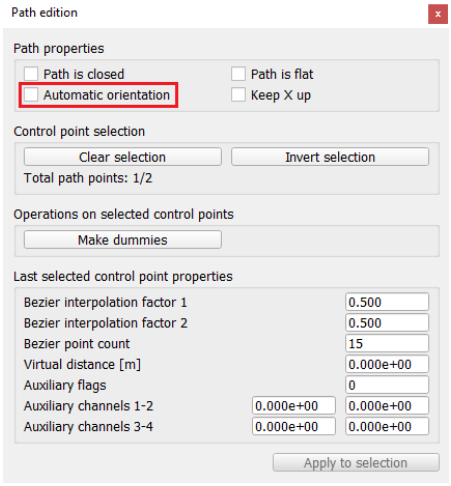
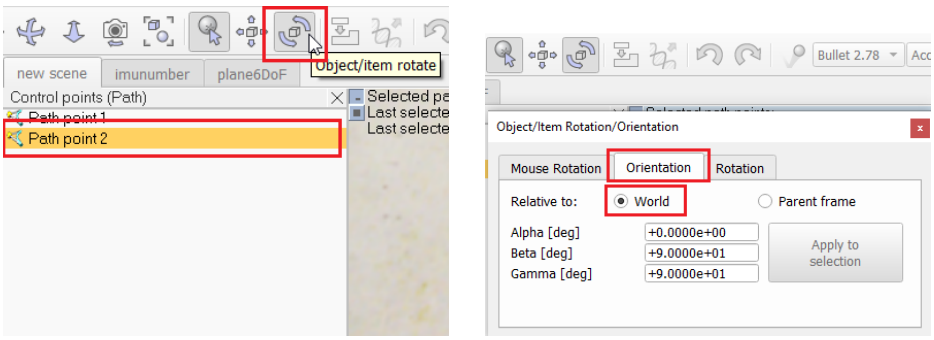


Figure 4.8: Toggle Autometic Rotation.

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.

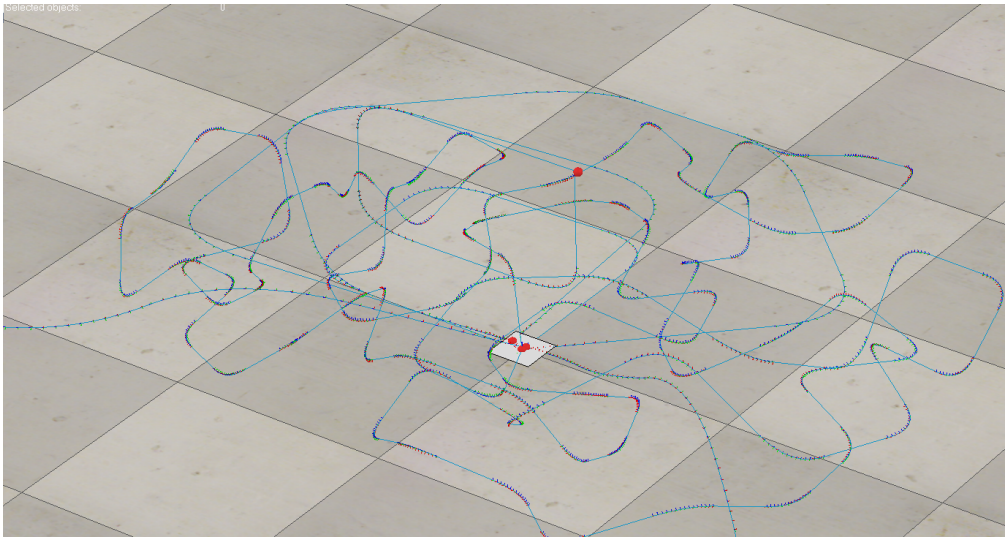


Figure 4.10: Final simulation path.

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

4.1.5 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.

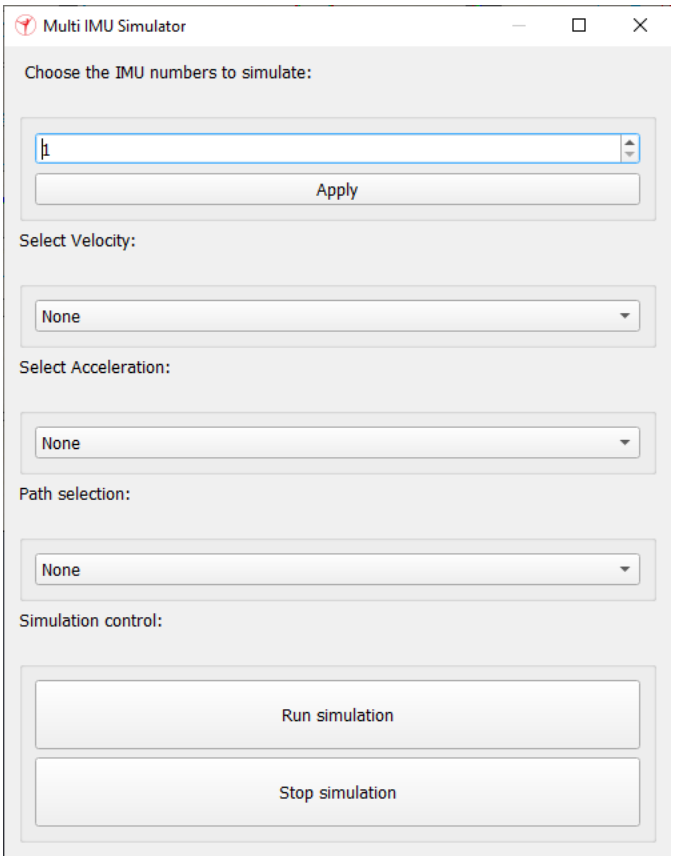


Figure 4.11: Simulation control panel.

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

4.1.5.1 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.

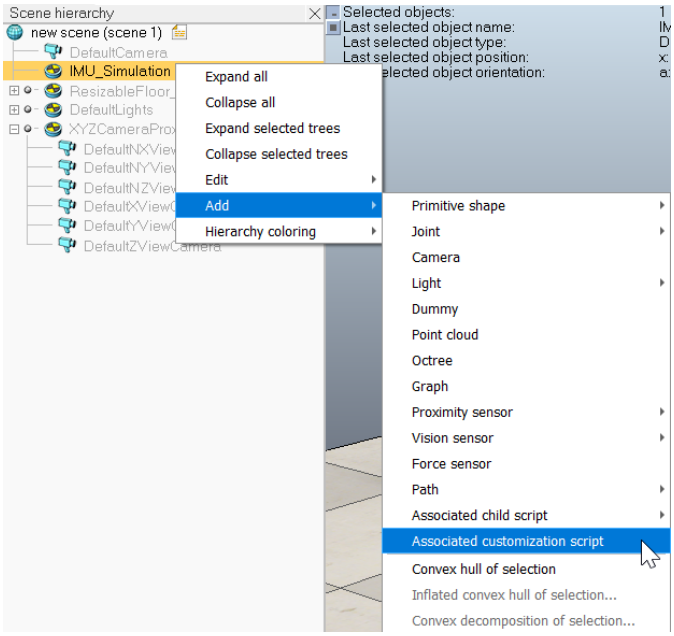


Figure 4.12: Customizing the script to create the control panel.

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

4.1.6 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.

4.1.6.1 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

4.1.6.2 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

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.

5.1 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.

5.1.1 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. The CPU must be liberated from use from different applications while the information assortment process is going on. 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.

6 Simulation

6.1 Section

Citation test [**latex**].

6.1.1 Subsection

See Table 7.1, Figure 7.1, Figure 7.2, Figure 7.3.

Table 6.1: An example for a simple table.

A	B	C	D
1	2	1	2
2	3	2	3

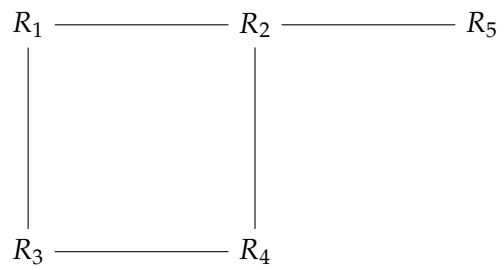


Figure 6.1: An example for a simple drawing.

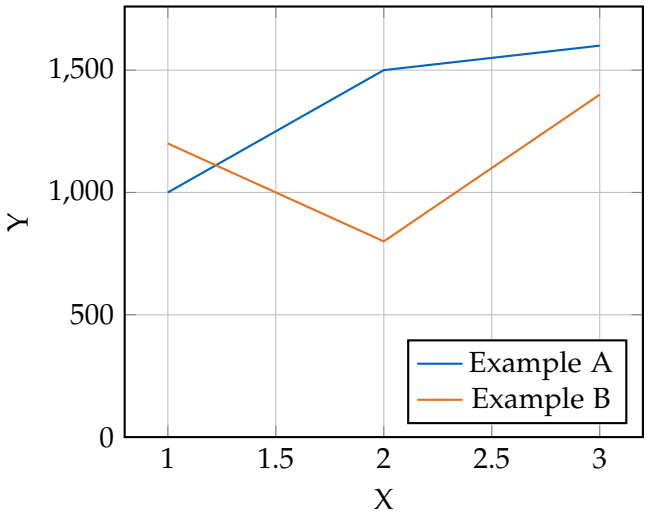


Figure 6.2: An example for a simple plot.

```
SELECT * FROM tbl WHERE tbl.str = "str"
```

Figure 6.3: An example for a source code listing.

7 Experiment Results and Comparison

7.1 Section

Citation test [**latex**].

7.1.1 Subsection

See Table 7.1, Figure 7.1, Figure 7.2, Figure 7.3.

Table 7.1: An example for a simple table.

A	B	C	D
1	2	1	2
2	3	2	3

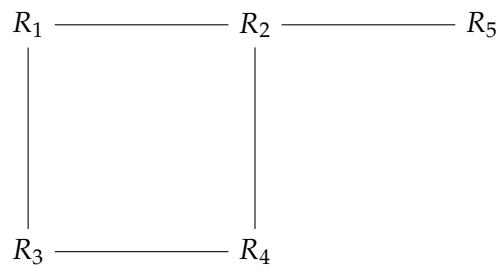


Figure 7.1: An example for a simple drawing.

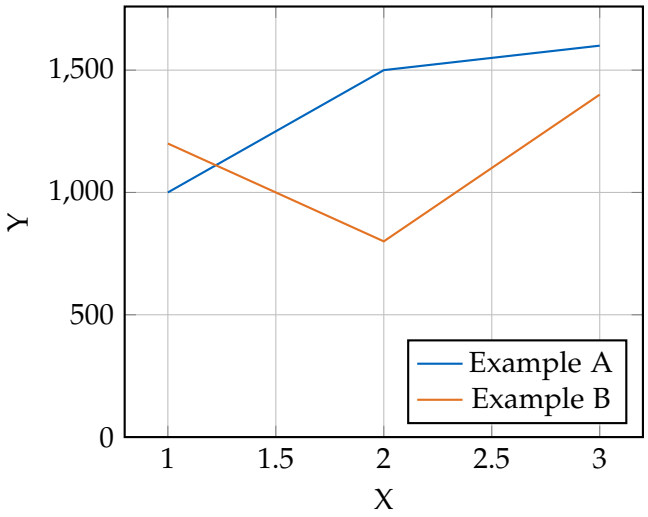


Figure 7.2: An example for a simple plot.

```
SELECT * FROM tbl WHERE tbl.str = "str"
```

Figure 7.3: An example for a source code listing.

List of Figures

List of Tables