Perception Methods For Speed And Separation Monitoring Using Time-of-Flight Sensor Arrays

Sarthak Arora
sa9472@rit.edu

Follow this and additional works at: https://scholarworks.rit.edu/theses

Recommended Citation

This Thesis is brought to you for free and open access by RIT Scholar Works. It has been accepted for inclusion in Theses by an authorized administrator of RIT Scholar Works. For more information, please contact ritscholarworks@rit.edu.
Perception Methods For Speed And Separation Monitoring Using Time-of-Flight Sensor Arrays

by

Sarthak Arora

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Electrical and Microelectronic Engineering

Supervised by

Dr. Ferat Sahin
Department of Electrical and Microelectronic Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
December 2019

Approved by:

Dr. Ferat Sahin, Professor
Thesis Advisor, Department of Electrical and Microelectronic Engineering

Dr. Gill Tsouri, Professor
Committee Member, Department of Electrical and Microelectronic Engineering

Dr. Sohail A. Dianat, Professor
Department Head, Department of Electrical and Microelectronic Engineering
Thesis Release Permission Form

Rochester Institute of Technology
Kate Gleason College of Engineering

Title:

Perception Methods For Speed And Separation Monitoring Using Time-of-Flight Sensor Arrays

I, Sarthak Arora, hereby grant permission to the Wallace Memorial Library to reproduce my thesis in whole or part.

Sarthak Arora

Date
Dedication

I dedicate this work to my parents, my wife and my sister for their immeasurable and undying love and support.
Acknowledgments

I’d like to begin by thanking Dr. Ferat Sahin for his guidance and for putting his belief in me as his graduate student.

When I began my journey to pursue my master’s thesis, I was extremely fortunate to have acquainted with Shitij P. Kumar and Celal Savur, who were Doctoral Candidates at that time. Working with them, empowered me to equip myself with the ability to see this work through. I had the opportunity to be a part of their ongoing research works from which this work stemmed out. I am extremely grateful for their guidance and support as senior graduate students. A special thanks goes to Shitij for granting me the opportunity to partner with him on one of his leading research works, without his guidance this work would have been incomplete.

Last but not least; no words would suffice to express my gratitude to my wife who emboldened me with her wisdom, love and support during countless hard days. I’d also like to thank my mother, father and sister.
Abstract

Perception Methods For Speed And Separation Monitoring Using Time-of-Flight Sensor Arrays

Sarthak Arora

Supervising Professor: Dr. Ferat Sahin

This work presents the development of a perception pipeline to passively track the partial human ground pose in the context of human robot collaboration. The main motivation behind this work is to provide a speed and separation monitoring based safety controller with an estimate of human position on the factory floor. Three time-of-flight sensing rings affixed to the major links of an industrial manipulator are used to implement the aforementioned. Along with a convolutional neural network based unknown obstacle detection strategy, the ground position of the human operator is estimated and tracked using sparse 3-D point inputs. Experiments to analyze the viability of our approach are presented in depth in the further sections which involve real-world and synthetic datasets. Ultimately, it is shown that the sensing system can provide reliable information intermittently and can be used for higher level perception schemes.
List of Contributions

Primary Contributions:

• Implementation of a perception pipeline for a bespoke range sensing system.

• A study of sensing coverage during human robot collaboration to analyse the sensing setup.

• Publications


Secondary Contributions:

• Implementation of a fast data association algorithm.

• Streamlining of software used in previous research works into a leaner version software application.
# Contents

Dedication ................................................................. iii

Acknowledgments ......................................................... iv

Abstract ................................................................. v

List of Contributions ................................................... vi

1 Introduction .......................................................... 1

2 Background and Related Work ........................................ 3
  2.1 Sensing .......................................................... 3
     2.1.1 Robot Perception ......................................... 3
     2.1.2 Sensors .................................................... 6
     2.1.3 Forward Kinematics ...................................... 6
     2.1.4 Time-of-Flight Sensors .................................. 8
     2.1.5 Sensor Model ............................................. 9
     2.1.6 Point Cloud Representation .............................. 10
  2.2 Human Robot Collaboration ....................................... 11
     2.2.1 Shared Workspaces ...................................... 12
     2.2.2 Speed and Separation Monitoring ....................... 12
  2.3 Estimation ........................................................ 14
     2.3.1 Multilayer Perceptron Networks ......................... 14
     2.3.2 Convolutional Neural Networks ......................... 16
  2.4 Tracking ........................................................ 16
     2.4.1 Linear Kalman Filter .................................... 17
     2.4.2 Particle Filters ......................................... 18

3 Methodology .......................................................... 19
  3.1 Sensor System .................................................... 19
  3.2 Data Processing .................................................. 20
List of Tables

2.1 A snapshot of the datasheet provided by STMicroelectronics for the VL53L1X sensor. .................................................. 10
4.1 Network Performance on the real world dataset during $\tau_{\text{robot}}$ and $\tau_{\text{human}}$. . . 47
4.2 Confusion Matrix for the Network Performance on the Real World Dataset. 47
4.3 Network Performance on the synthetic dataset during $\tau_{\text{robot}}$ and $\tau_{\text{human}}$ in simulation. ................................. 48
4.4 Confusion Matrix for the Network Performance on the Synthetic Dataset. . 48
4.5 This table shows the RMSE values for the plots shown before. .................. 60
List of Figures

2.1 An example of a perception pipeline inspired by [1]. ....................... 4
2.2 On the left, is shown a position encoder, AMT Modular Encoder made by CUI Devices [2]. The SICK LMS5XX 2D Lidars [3] are shown on the right. 6
2.3 An illustration of a serial link manipulator with four joints and links. .... 8
2.4 On the left, is shown a VL53L1X module from the STMicroelectronics Datasheet. The time-of-flight is shown on the right. ......................... 9
2.5 On the left, is shown a pointcloud representation of the Stanford Bunny. On the right, an octree representation of the Bunny is shown using the Point Cloud Library. .................................................. 11
2.6 An image illustrating the workspace overlap of the human and a UR10 robot manipulator. The workspace of UR10 is represented by the green colored volume and the human’s workspace is shown in blue. ............... 13
2.7 An image illustrating the regions used in speed and separation monitoring during robot operation. The green region represents the area on the ground which has no effect on robot operation speed. Yellow region is for reduced robot speed whereas red for a safety rated stop. Please note that these regions are only shown for the purpose of illustration and are not drawn to scale. .......................................................... 13
2.8 An illustration of a multi-layer perceptron modified and adapted from Petar Veličković’s GitHub repository. An exploded view of one neuron is shown along with its operations. .............................................. 15
2.9 An illustration of a convolution operation adapted from Petar Veličković’s GitHub repository. A kernel K is convolved over the input image I and produces an output as shown. .............................................. 17
3.1 An illustration of the proposed perception pipeline which is explained further. 19
3.2 The top view schematic diagram of a sensor ring is shown on the left where each sensor unit is represented by s. The image on the right is adapted from [4], the sensor ring mounting locations are shown. ......................... 20
3.3 An illustration of the time synchronization process to achieve a common time axis across multiple streams. ................................. 22
3.4 An illustration of the time sensing ring configuration for the UR10 robot manipulator. The $x' - y' - z'$ frame represents $P_{\text{Observation}}$.

3.5 An illustration of the time series buffering using a double ended queue. $s_t$ represents an incoming sample at time $t$.

3.6 A point cloud shown on the left with its corresponding bird’s eye view on the right. The $z$ values are mapped to rgb values and are shown as a colormap.

3.7 An illustration of the cylindrical link bound by an inflation volume. The point is shown with respect to the cylinder’s reference frame, $x_c - y_c - z_c$.

3.8 An illustration of the 2D convolutional network which takes an bird’s eye view input images of size $21 \times 21$ and outputs 24 probabilities for each sensor on the three rings of the robot.

4.1 Pictures showing the experiment setup. The simulation instance is shown on the left and the real world setup is shown on the right. The workspace is occupied by a human operator while the robot is performing a task.

4.2 An illustration showing the hardware and the software components in a block diagram fashion.

4.3 A picture showing the experiment setup for a single unit VL53L1X sensor. The wooden plank is the obstacle and the sensor is encircled in red. The distance between the obstacle and sensor is 600mm.

4.4 The plot shows an error graph for the VL53L1X for a baseline of 600 mm.

4.5 The plot shows error distribution for the VL53L1X for a baseline of 600 mm.

4.6 The plot shows an error graph for the VL53L1X for a baseline of 600 mm with the obstacle translated by 1 inch to the right side of the optical axis.

4.7 The plot shows error distribution for the VL53L1X for a baseline of 600 mm with the obstacle translated by 1 inch to the right side of the optical axis.

4.8 The plot shows an error graph for the VL53L1X for a baseline of 600 mm with the obstacle translated by 2 inches to the right side of the optical axis.

4.9 The plot shows error distribution for the VL53L1X for a baseline of 600 mm with the obstacle translated by 2 inches to the right side of the optical axis.

4.10 The plot shows an error graph for the VL53L1X for a baseline of 600 mm with the obstacle translated by 1 inch to the left side of the optical axis.

4.11 The plot shows error distribution for the VL53L1X for a baseline of 600 mm with the obstacle translated by 1 inch to the left side of the optical axis.

4.12 The plot shows an error graph for the VL53L1X for a baseline of 600 mm with the obstacle translated by 2 inches to the left side of the optical axis.
4.13 The plot shows error distribution for the VL53L1X for a baseline of 600 mm with the obstacle translated by 2 inches to the left side of the optical axis.

4.14 The plot the trajectory of the human while performing an arbitrary task while working with the robot. The robot is located at the origin of the graph at (0,0).

4.15 The plot of distance versus number of detections performs by the sensors.

4.16 A bar graph showing number of occurence of number of points observed per point cloud when the human is occupying the robot workspace.

4.17 A point cloud shown on the left with its corresponding bird’s eye view on the right. The \( z \) values are mapped to rgb values and are shown as a colormap. These obtained by applying the output of the network to the input pointcloud.

4.18 This plot shows the network output overlayed on top of the human trajectory.

4.19 This plot shows the ground truth for the network overlayed on top of the human trajectory. The ground truths are generated from a physics engine instance.

4.20 A point cloud shown on the left with its corresponding bird’s eye view on the right. The \( z \) values are mapped to rgb values and are shown as a colormap. These obtained by applying the output of the network to the input pointcloud.

4.21 This plot shows the network output overlayed on top of the human trajectory. These results show the inference on the synthetic dataset.

4.22 This plot shows the ground truth for the network overlayed on top of the human trajectory. The ground truths are generated from the physics engine instance.

4.23 This plot shows a 2D density histogram of the training data used for the human position. The hollow region in the middle is the part of the volume that cannot be accessed by the human operator as it is occupied by the robot and other entities.

4.24 This plot shows the predefined ground truth trajectory for the human operator in a simulation overlayed by the centroid of the points observed by the sensor.

4.25 This plot shows the predefined ground truth trajectory for the human operator in a simulation overlayed by the kalman filter output.

4.26 This plot shows the predefined ground truth trajectory for the human operator in a simulation overlayed by the particle filter output.
4.27 This plot shows the predefined ground truth trajectory for the human operator in a simulation overlayed by the MLP output of the points observed by the sensor. .................................................. 57
4.28 This plot shows the predefined ground truth trajectory for the human operator in a simulation overlayed by the kalman filter output. ................................. 58
4.29 This plot shows the predefined ground truth trajectory for the human operator in a simulation overlayed by the particle filter output. ................................. 59
Introduction

The advent of collaborative robots has enabled robots and humans to work in close proximity of each other. This has invoked a need for safety protocols and standards for the human operator working on factory floors in an industrial setting. To ameliorate the problem of working with robots confined in cages, ISO/TS 15066 [5] was introduced in the year 2016, which dictates the implementation of safety controller using speed and separation monitoring (SSM). SSM is fundamentally the continuous monitoring of human robot separation distance in order to control the robot operation speed. However, the controller can also benefit from human pose information which can be used to estimate the ground speed of the operator wrt to the robot and can provide for better control.

The motivation behind this work stems out from the following:

- **Justifying the use of a sensing system that is completely affixed to the robot’s body:** Traditionally, sensors that are situated in the robots environment are used, these sensors can be a suite of multi-modal sensors or can be bigger sensing system. For brevity, the aforementioned is referred to as off-robot sensors as shown in [6]. The use of off-robot sensors comes with certain challenges such as:
– Volume Constraints

The volume that entails the robot workspace must be large enough to posses an off-robot sensing setup to satisfy thresholds such as minimum and maximum observable ranges for continuous monitoring.

– System Configuration and Placement Constraints

The process of making decisions on where to place the sensors to maximize observability of the workspace and at the same time configuring and calibrating the off-robot sensing system can be challenging and time consuming.

The joint ramification of the above two challenges implies that a room or a factory must have area that is large and unoccluded for reliable sensing. Therefore, evaluating the use of on-robot sensors may provide for a solution that can solve these problems.

• The need for leveraging information that can enable better SSM based control for HRC:

Analysing the information provided by on-robot sensors to extract pose information can be for trajectory estimation thereby enabling SSM to be extended to motion planning. Also, the ground speed estimation of the human operator can provide a smoother and safer controller output.

An in-depth explanation of SSM is discussed in Section 2.2.2.
Background and Related Work

This chapter attempts to provide a primer on the techniques and algorithms used in this work. Each section briefly explains an algorithm or a technique.

2.1 Sensing

In the realm of robotics, most problems can be classified under perception, planning and control. However, for an end-to-end robotic system, the problem of perception must be dealt with first, as every robotic system primarily comprises of sensors and actuators. Applying techniques and algorithms to process raw sensing data is what embodies perception.

2.1.1 Robot Perception

According to [1], perception is defined as a process that entails raw sensing data acquisition, processing and transformation to build a world model. In other words, perception is a multistaged process which is used to build an in-situ representation of the environment in which the robot is operating. This representation can never be ideal or complete as compared to real world as the sensing data always holds limited information about the world. A high level block diagram (Figure 2) is shown below to portray the idea.

As shown in the figure, the sensing information can originate from different modalities and devices. The raw data has to preprocessed, transformed and features must be extracted
Figure 2.1 An example of a perception pipeline inspired by [1].

for relevant information. As mentioned in [1], there can be a myriad of ways in which this pipeline can be implemented. A general explanation of each block is given further:

- **Acquisition**

  Acquisition is the first stage in the pipeline and is also one of the most important stage. Typically, there can be many sensors in a robotic system providing data at different sampling frequencies. A lot of times, the data arrives asynchronously, in order to draw useful information from all the sensors, it is important to synchronise the incoming data, temporally. For obtaining a complete observation of the robot state and its environment, time synchronization of incoming data is vital.

- **Preprocessing**

  Preprocessing is used for noise filtering or artifact removal from the sensor data. Another key preprocessing step is to also build a data buffer which can be really useful for feature extraction techniques which require more than one time step of data and observations.

- **Transformation**

  There is no specific technique for transformation, as it is completely case specific. Transformation essentially means to apply a mapping to an input space to obtain a
specific output space. For example, applying transformations on polar data to obtain cartesian data is commonly done when working with 2D laser scanners.

- **Feature Extraction**

  Feature extraction implies the extraction of relevant information from the data that is best suited to building an efficient representation of the robot state and the environment. Feature extraction can also be extremely algorithm specific, a lot of modern perception pipelines use machine learning techniques which inherently perform feature extraction on the data. Usually, a lot of feature extraction techniques are applied to computer vision data. Some examples of such algorithms are SIFT [7], SURF [8] and ORB [9].

- **Data Association**

  Data association is a type of an assignment problem. It deals with the problem of assigning the features computed during feature extraction to entities that exist in the robot environment.

- **Estimation**

  Estimation comprises of models of dynamical systems that propagate states further in time and leverage current measurements. Estimation can usually posses generative or discriminative models to produce estimates of quantities in the system.

- **Model Building**

  Model building is usually the final step in the pipeline and represents the state of the environment and the robot. An example of model building is to build maps of the
world using sensing data. The perception pipeline is highly iterative by design and
requires constant feedback in the form of previous data.

2.1.2 Sensors

[1] classifies sensors into two categories:

- **Proprioceptive Sensors**
  
  These sensors provide information about robot’s intrinsic quantities. For example,
  positional encoders provide a measure of rotation of a robot’s joint angles.

- **Exteroceptive Sensors**
  
  These sensors provide information about robot’s environment. For examples, a 2D
  laser scanner uses the time-of-flight principle to calculate the distance to the obstacles
  in the robot environment.

![Figure 2.2](image)

**Figure 2.2** On the left, is shown a position encoder, AMT Modular Encoder made by CUI Devices [2]. The
SICK LMS5XX 2D Lidars [3] are shown on the right.

2.1.3 Forward Kinematics

Forward kinematics provide a one-to-one mapping from joint angles to cartesian coordi-
nates of the end effector in a robotic manipulator. Forward kinematics are computed using
successive matrix multiplications. To understand forward kinematics, the concept of frame transformations is important. A frame transformation implies is represented by rotation matrices and translation vectors. The general form for computing the cartesian pose of an end effector in a serial link manipulator is given by the following relation:

$$\text{World}_T^{\text{End Effector}} = \text{World}_T^{J_i} \cdot J_{i+1} \cdots J_n \cdot T_{\text{End Effector}}$$ \hspace{1cm} (2.1)

The general form of a transformation of the current frame with respect to a preceding frame in the manipulator is given by:

$$\text{Preceding Frame}_T^{\text{Current Frame}} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix}_{4x4}$$ \hspace{1cm} (2.2)

In equations 3.1 and 3.2:

- $J_i$ is the $i^{th}$ joint in the manipulator with $n-1$ joints.
- $R$ is the rotation matrix and $t$ is the translation vector.

- $\{\text{World}\}$ is a reference frame for the $\{\text{End Effector}\}$ frame.
2.1.4 Time-of-Flight Sensors

Time-of-flight sensors are a class of active sensors that emit energy in the form of light or sound to compute the time taken for one trip. The time is then halved and multiplied by the speed of light or sound depending on the sensor. For many years ultrasonic sensors [10] have been used in robots for tasks such as mapping and localization, [11], [12], [13], [14], [13] and [15]. Recently, many laser based single unit lidars such as VL53L0X [16] and VL53L1X [17] have been introduced due the rapidly reducing price of sensing technology.
Time-of-flight Principle and VL53L1X

The VL53L1X possesses an emitter and a receiver which are essentially a vertical cavity surface emitting laser (VCSEL) and a single photon avalanche diode (SPAD), respectively. The VCSEL emitter emits a beam of photons at 940nm (Class 1 Laser) whereas the SPAD receiver is an array grid with a lens. The working principle of the sensor is illustrated below:

![Figure 2.4](image)

The time-of-flight is represented by the delay between the pulses caused at emitter and the receiver. The principle can be explained by a simple equation:

\[
\text{Distance (in picometers)} = \frac{\Delta t}{2} \times 2.99792 \times 10^{20}
\]  

(2.3)

Some of the important specifications of the VL53L1X sensor as given by the manufacturer [17] are shown in Table 2.1.

2.1.5 Sensor Model

This section describes an approach to develop a theoretical model of how the VL53L1X performs sensing and how multiple VL53L1X units are daisy chained in ring fashion. According to the manufacturer information [17], the sensor works on the pinhole camera
principle such that the SPAD receiver acts like a camera sensor. The SPAD array can be imagined as a 2 dimensional matrix where each cell provides the measured signal reflectance. Since the manufacturer allows the number of cells used to perform a reading, the field of view of reception can be tweaked. The sensor usually performs a reading in a 27 degree conical field of view which is loosely similar to ultrasonic sensor field of view of the main lobe. A theoretical model of this sensor is created inside a physics engine by using ray-casting. Where each time an obstacle intersects the field of view cone, a ray is cast and the distance is measured. Using this simple approach, a theoretical model is created.

2.1.6 Point Cloud Representation

A point cloud is an un-ordered data structure that stores 3-dimensional points as x,y & z coordinates. When using a 3D lidar or a depth camera, the raw point clouds produced by these sensors are very dense in nature and can easily explain the spatial structure of the observed objects. Creating a point cloud usually requires applying frame transformation on 3-D point to obtain points with reference to a common frame in the homogeneous space. Processing point clouds can be very expensive and should be used with libraries such as
[18] for efficient data processing algorithms. For efficient processing, point clouds are usually stored in tree based data structures as compared to linked-list based data structures as that drastically reduces the algorithmic time complexity. An example of a point cloud is show below.

**Figure 2.5** On the left, is shown a pointcloud representation of the Stanford Bunny. On the right, an octree representation of the Bunny is shown using the Point Cloud Library.

---

# 2.2 Human Robot Collaboration

Human robot collaboration (HRC) can be defined as a process to tackle some challenges that occur during the workspace overlap between a robot and a human operator. Until very recently, robots have operated in safety cages and have to be completely turned off for a human operator to intervene the robot’s task. As the collaborative robots were introduced in the industry, the use of cages has been eliminated. Collaborative robots allow human operators to operate in very close proximity while maintaining the safety and task execution. From a broad perspective, human robot collaboration addresses three major challenges as
mentioned in [19]:

- Safe Interaction for HRC
- Intuitive Interfaces for HRC
- Design Methods for HRC

### 2.2.1 Shared Workspaces

The idea of a shared workspace is central to HRC. When a robot and human perform a joint task, their workspace is bound to overlap. An illustration of this shown further.

### 2.2.2 Speed and Separation Monitoring

Speed and Separation Monitoring (SSM) is the third collaborative operation scenario as specified in ISO TS 15066:2016 [5]. In SSM, the robot operation speed is a function of the separation distance between the human operator and the robot. The standard provides the equation to compute the minimum protective separation distance. The equation is given and explained as shown in [20]:

\[
d_C(t) \geq V_H(t)T_R + V_H(t)T_S + V_R(t)T_R + B + C + Z_s + Z_r \tag{2.4}
\]

The key components in the equation are \(V_H(t)\) & \(V_R(t)\) that represent the speed of the human operator and the robot, respectively. \(T_S\) & \(T_R\) represent the reaction times of sensing system and the robot, respectively. \(Z\) represents the uncertainty where \(s\) & \(r\) are for human and robot pose. \(B\) is the maximum stopping distance for the robot and \(C\) is an arbitrary constant used as cushion. An illustration of SSM is shown further.
**Figure 2.6** An image illustrating the workspace overlap of the human and a UR10 robot manipulator. The workspace of UR10 is represented by the green colored volume and the human’s workspace is shown in blue.

**Figure 2.7** An image illustrating the regions used in speed and separation monitoring during robot operation. The green region represents the area on the ground which has no effect on robot operation speed. Yellow region is for reduced robot speed whereas red for a safety rated stop. Please note that these regions are only shown for the purpose of illustration and are not drawn to scale.
2.3 Estimation

The aim of this section is to discuss methods that provide an estimate for a quantity using machine learning techniques. We discuss two types of algorithms here; supervised and unsupervised. One of the the primary aims of estimation is to find $\hat{y}$ such that the error between a ground truth $y$ and $\hat{y}$ is minimized using parameterization. The objective can be formalized as:

$$\arg\min_{w, b} L(y, \hat{y})$$  \hspace{1cm} (2.5)

$$\text{where } \hat{y} = f(w, b)$$ \hspace{1cm} (2.6)

By minimizing the error between $\hat{y}$ and $y$, we are essentially maximizing the probability $P(y | x)$ where $x$ is the sample corresponding to the ground truth $y$ in a training set.

2.3.1 Multilayer Perceptron Networks

A multilayer perceptron network is a type of artificial neural network, it is usually referred to as a feed forward network which implies that the connection between the units in the network are non-cyclical and sequential only. The fundamental idea behind a multilayer perceptron is to minimize a loss function by training it on a training set $\{X, Y\}$ where $X$ represents the set of samples $\{x_1, \ldots, x_n\}$ and $Y$ represents the set of ground truths corresponding to each sample in $X$: $\{y_1, \ldots, y_n\}$. The best way to represent and visualize a multilayer perceptron is by a graph, where nodes are known as units and edges are known as connections. Each connection has a scalar weight $w_{i,h}$ and a bias $b_{i,h}$ where $h$ represents a layer index and $i$ represents the connection index. Each layer can have an arbitrary number of units.
depending upon the network designer and the performance. Due to the fully connected nature of the graph, where the outermost layers are known as input & output layers and the inner layers are known as hidden layers, each unit in a layer is connected to every unit in its preceding and subsequent layers.

**Figure 2.8** An illustration of a multi-layer perceptron modified and adapted from Petar Veličković’s GitHub repository. An exploded view of one neuron is shown along with its operations.

The operations performed in a multilayer perceptron can be broken down into a series of linear combinations which are subjected to non-linear functions. The output for a network with a single hidden layer can be represented as:

$$\hat{y} = g(W_{i+1}f(W_{i}x_{n} + b_{i}) + b_{i+1})$$  \hspace{1cm} (2.7)

Where $g$ and $f$ represent non-linearities that are applied to the linear combination. Note that $g$ is an identity function in most cases. Once the forward is computed the error or loss $L$ between $\hat{y}$ and $y$ is minimized using the objective shown in equation 3.5. The process of minimization is achieved using backpropogation [21] of error where the error is differentiated with respect to the parameters in the network that are given by $W_{i}$ and $b_{i}$. 
Gradient descent is then used update the parameters of the network to satisfy the objective in eq 3.5. The following operations are used to minimize the loss using a hyperparameter $\alpha$:

$$W_i \leftarrow W_i - \alpha \frac{\delta L(y, \hat{y})}{\delta W_i}$$

$$b_i \leftarrow b_i - \alpha \frac{\delta L(y, \hat{y})}{\delta b_i}$$

### 2.3.2 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are another class of feedforward artificial neural networks. The fundamental idea behind CNNs is the use of parameter sharing. As the dimensionality of the sample increased more parameters are required to learn the representation of the data. However, if an MLP based approach is used, the number of parameters drastically increase. To reduce the required parameters, a kernel is used in a sliding window fashion over the input. The elements of the kernel are weights and biases that are then subjected to non-linearities. CNNs [22] were originally designed for image data. For example, a grayscale image is 2-dimensional and possesses spatial information, a kernel is used to convolve over the image where each element in the kernel is a parameter, as a result of which a smaller feature map is obtained and is flattened and passed on to a smaller multilayer perceptron network for classification or regression purposes. An illustration of a convolution operation used in CNNs is shown below:

### 2.4 Tracking

Tracking implies the use of state space based bayesian techniques to predict and update the next state given a measurement. As shown in [15], there are different types of bayesian
filters but use the same underlying concept. Traditionally, a sensor measurement is referred to as \( z \) and a state is referred to as \( x \). The posterior \( P(x \mid z) \) can be calculated by a simple bayesian update:

\[
P(x \mid z) = \frac{P(z \mid x) P(x)}{P(z)}
\]

(2.8)

The denominator in eq 3.8 is usually referred to as a normalization term, the conditional \( P(z \mid x) \) is called the likelihood and \( P(x) \) is called the prior.

### 2.4.1 Linear Kalman Filter

The Kalman Filter was introduced in [23], which has influenced decades of research. As specified in [15], the kalman filtering algorithm is defined by the following set of equations:

**Predict**

\[
\begin{align*}
\hat{x}_k^- &= A\hat{x}_{k-1} + Bu_k \\
\hat{P}_k^- &= A\hat{P}_{k-1}A^T + Q
\end{align*}
\]

(2.9)

**Update**

\[
K_k = \frac{P_k^-H_k^T}{H_P^kH_k^T + R_k}
\]

\[
\hat{x}_k = \hat{x}_k^- + K_k(z_k - H_k\hat{x}_k^-)
\]

(2.10)

\[
P_k = (I - K_kH_k)P_k^-
\]
The equations are grouped into predict and update steps where $K$ is the kalman gain, $P$ is covariance $Q$ is process noise, $H$ is the measurement function and $A$ is the state transition function. $B$ and $u$ allow for taking in control inputs.

2.4.2 Particle Filters

Particle Filters are used to model problems with non-linearities and is a type of bayesian filter. In a particle filter the belief is represented by particles where each particle has a weight. It must be noted that the weights are normalized. During the update step particles closes to the measurement form the posterior. In [24], an in-depth review of modern particle filter techniques is shown. A very prominent particle filter algorithm is an importance sampling based approach called sampling importance resampling as shown in [24]. During the predict step in the filter, each particle is subjected to a state transition function incorporating noise for a robust estimate.
Methodology

This chapter discusses the methodology.

Figure 3.1 An illustration of the proposed perception pipeline which is explained further.

3.1 Sensor System

As mentioned earlier, this work utilizes three arrays of time of flights sensors. This section discusses the sensing setup that is used to implement this work.

A time-of-flight ring is implemented by daisy-chaining eight single unit lidars (VL53L1X [17]). Three of such sensor rings are mounted on a robot. Each sensing ring output eight distance readings in millimeters. The robot used in this work is the Universal Robots UR10
The sensing rings are mounted on each link of the robot such that the transformation between the ring and the link center is zero. The links used are Base, Elbow and Tool. The tool link is the smallest and extends to an attachment end for using different types of end effectors. An illustration of this setup is shown below:

**Figure 3.2** The top view schematic diagram of a sensor ring is shown on the left where each sensor unit is represented by \( s \). The image on the right is adapted from [4], the sensor ring mounting locations are shown.

The rings are designed in such a way that there is angle offset between every two sensors of 45 degrees and each sensor monitors a volume of 27 degrees. Further details are discussed in the forthcoming sections.

### 3.2 Data Processing

The data processing stage involves preprocessing techniques such as noise filtering and thresholding. Each sensing ring has dedicated micro-controller that interacts with individual sensor units in the rings using the I2C protocol. The micro-controller polls the sensors
and assembles a message consisting of eight raw distance values and forwards the data over serial to a ROS Master [26]. Once the data from the three ring is received by the central node, preprocessing steps are performed. The data is first subjected to a first order low pass filter for smoothing, as the sensor also provides some auxiliary signal intensity values, by experimentation, these values were thresholded to get best output. Finally, the distance values are capped at 1.3 meters and any reading beyond 1.3 meters is treated as an arbitrarily high value.

As the data from the sensors is assimilated into the system, data from a motion capture system, OptiTrack [27] and the data from the proprioceptive sensors in the robot that provide the joint position is also received. As the data from different streams is ingested in the system, it is quite evident that the data for each stream will be at different sampling frequencies. To monitor that complete state of the system that includes the environment and the robot, the data must be temporally synchronized; in other words, all the sensor streams must have a common time axis in order to monitor the state $S_t$ at time $t$ of the system. The state of the system can formalized as:

$$S_t = \{z_{\text{sensor ring}}^i, z_{\text{joint positions}}^i, z_{\text{motion capture}}^i\} \quad (3.1)$$

In eq 3.1, $i \in (1,2,3)$, where $i$ represents the sensing ring index. $z$ represents the measurement made by each sensing system. To achieve time synchronization over all the incoming sensor streams, a time approximation [28] policy was used which synchronized streams at different sampling frequencies to the lowest sampling frequency in the system. The time-of-flight rings operate at 16 hz approximately whereas the joint position encoders and the
optitrack system operate at 120 hz. An illustration of this process is shown below:

**Figure 3.3** An illustration of the time synchronization process to achieve a common time axis across multiple streams.

As the aforementioned describes the approach developed for a real world setup, a simulation based setup using V-REP [29] was modelled to represent the environment and the robot. The simulation was used to test and generate analogous data as the simulator did not suffer with the problem of asynchronous data stream along with variable sampling frequencies. In the simulation, the $S_T$ can be directly sampled without time synchronization.
After the $S_r$ is formed, the data is then ready for higher level processing as explained in further sections.

### 3.2.1 Kinematic Transformations

The kinematic transformations are used to transform 1-dimensional distance values to homogeneous coordinates. The idea behind this approach is to augment the kinematic chain of the robot with the sensor rings. As each rings is completely aligned with the robot links, the rings are subjected to the same transformation that the links go through, as a result of this, distance readings provided by each sensor unit within a ring can be transformed to obtain a homogeneous coordinate. The transformation is shown below:

$$\begin{align*}
\text{World} P_{\text{Observation}_{s,r}} &= \text{World} T_{\text{Sensor}_{s,r}} \cdot \begin{bmatrix}
\text{distance}_{s,r} \\
0 \\
0 \\
1
\end{bmatrix},
\end{align*}$$

(3.2)

Where $s \in \{1,2,3,4,5,6,7,8\}$ & $r \in \{1,2,3\}$, $s$ is the sensor index in the ring index, $r$. $P$ is the homogeneous coordinate and $T$ is the transformation. The $\text{distance}_{s,r}$ reported by the $s_{th}$ sensor in the $r_{th}$ ring is then post multiplied by the transformation matrix of the respective sensor. It must be noted that the transformation matrix of each sensor unit in its respective ring is pre-computed by using the robot’s kinematic chain. As $P$ is calculated for each sensor unit in each ring, a total of 24 (8 sensors $\times$ 3 rings) homogeneous coordinates are collected. As the first three elements of each homogeneous coordinate consist of the cartesian coordinate with respect to a common reference frame, the elements can then be
extracted and used to assemble a point cloud. It must be noted that if the sensor reports a zero distance, its corresponding $P$ computation is skipped as it returns the sensor location. Another key challenge to keep in mind is that, the distance reported is originating from an obstacle in a 27 degree field of view which embeds error in the point cloud $PC_t$ as there is no information available about the other dimensions. To overcome this problem, only the distance reading that have the maximum confidence of being along the optical axis of the sensor are captured. Therefore, $PC_t$ is a point-cloud that represents the spatial observation made by all the sensing rings at time $t$ such that $PC_t$ is an unordered set of cartesian points extracted from homogeneous coordinated computed in Equation 3.2. An illustration of this is shown below:

**Figure 3.4** An illustration of the time sensing ring configuration for the UR10 robot manipulator. The $x' - y' - z'$ frame represents $WorldP_{Observation}$.
3.2.2 Buffering

Buffering is used to increase the richness of each sample. As $PC_t$ is extremely sparse in nature due to the design of the sensing rings. There are many times when $PC_t$ would contain no information. To overcome this, point cloud samples are stored in a circular buffer and a union of $p$ point clouds is calculated and a denser $PC_t$ is formed. Thereby, increasing the number of points per point cloud. An illustration of this is shown below:

**Figure 3.5** An illustration of the time series buffering using a double ended queue. $s_t$ represents an incoming sample at time $t$. 

![Buffering Diagram](image-url)
3.2.3 Bird’s Eye Views

Bird’s Eye Views (BEV) are essentially elevation maps that represent 3-dimensional information in form of images. The use of Bird’s Eye Images has been very popular in the challenge introduced by [30]. Works such as [31] and [32] use birds eye views as it has allowed researchers to directly feed point clouds as images to convolutional neural networks [22]. It can be seen from empirical results of various works performed on [30] that bird’s eye views are a successful approach for point cloud processing using deep learning [33]. An elevation map of $PC_t$ gives the bird’s eye view representation, which is essentially an image where each pixel location is the $x$ & $y$ coordinate and the pixel intensity is the $z$ coordinate. It must be noted that, if an elevation map of a dense point cloud is constructed, there can be some information loss as only the highest point in the point cloud is represented. However, in this approach, as $PC_t$ is much sparser, the information loss is ignored.

To compute a BEV, a region of interest (roi) was defined as a volume of 8 cubic meters ($2 \times 2 \times 2$), each 3D point in that roi was then plotted as a 2D point without its elevation on a 2D grid. The plotted pixel was then assigned an intensity by scaling the values of the corresponding point’s elevation between 0 & 255. The grid resolution was chosen to be 5 centimeters. An illustration of the BEV is shown in figure 3.6:
Figure 3.6 A point cloud shown on the left with its corresponding bird’s eye view on the right. The $z$ values are mapped to rgb values and are shown as a colormap.

3.3 Data Association

This section describes the data association algorithms implemented to assign an identity to each 3D point in $PC_t$. To deal with this assignment problem two approaches are proposed. To deal with this problem, the state $S_t$ obtained from the sensors is applied to simulation instance such that the simulation mimics the physical system. Apart from applying the data from $S_t$, the simulated sensor model for each unit $s$ in each ring $r$ is instantiated as $s'$ and $r'$. Therefore, as each reading in the physical system is performed at time $t$, a temporally corresponding reading using $s'$ and $r'$ is performed using ray casting. As the environment can be broken down as known and unknown obstacles where known obstacles are entities that are static and unknown obstacles are dynamic entities such as the human operator, the environment can be modelled in the simulation and the identities of obstacles can be obtained. This enables us to obtain the identity of the obstacle on which a ray is cast, which further enables us to identify known obstacles in the physical world with some approximation [20].
3.3.1 Self-Filtering Algorithm

The self filtering algorithm is a scan based approach [15], where the 3D points of a point cloud are projected into the world model inside the simulation instance. If the projected point lies inside an inflation region that bounds the robot body at time $t$, then the point is classified as known. Although this approach is intuitive and easy to implement the challenge arises when the number of known obstacles in the world increase. As each point has to be assessed against all known obstacles in the world model, it can be quite computationally expensive to do so. To overcome this problem, this algorithm only searches for points that might lie on the robot within an inflation region around it and the remaining points are identified using another approach given further. The algorithm is given below:

Procedure 3.1 Self-Filtering Algorithm

- **Data:** $PC_t$, $WT_{Base}^t$, $WT_{Elbow}^t$, $WT_{Tool}^t$
- **Result:** $PC_t^*$ - Filtered Point Cloud

<table>
<thead>
<tr>
<th>Procedure 3.1 Self-Filtering Algorithm</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Data: $PC_t$, $WT_{Base}^t$, $WT_{Elbow}^t$, $WT_{Tool}^t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Result: $PC_t^*$ - Filtered Point Cloud</td>
</tr>
<tr>
<td>Transforms $= (WT_{Base}^t, WT_{Elbow}^t, WT_{Tool}^t)$</td>
</tr>
<tr>
<td>While $n \leq \text{size(transforms)}$ do</td>
</tr>
<tr>
<td>While $p \leq \text{size}(PC_t)$ do</td>
</tr>
<tr>
<td>$[\text{Link } P_{\text{Observation}}]_n^{-1} \cdot [PC_t]_p$</td>
</tr>
<tr>
<td>If $\text{inflation}^\text{height} &lt; [\text{Link } P_{\text{Observation}}]_{3,1}$</td>
</tr>
<tr>
<td>If $[\text{Link } P_{\text{Observation}}]<em>{2,1}^2 + [\text{Link } P</em>{\text{Observation}}]_{2,1}^2 &lt; \text{inflation}^\text{radius}$</td>
</tr>
<tr>
<td>$[\text{Link } P_{\text{Observation}}] \leftarrow 0_{3x1}$</td>
</tr>
<tr>
<td>End</td>
</tr>
<tr>
<td>End</td>
</tr>
<tr>
<td>$PC_t^* \leftarrow PC_t$</td>
</tr>
<tr>
<td>End</td>
</tr>
</tbody>
</table>

A conceptual illustration is shown to explain how Procedure 3.1 works.
3.3.2 Stochastic Masks using Convolutional Neural Networks

A CNN [22] based approach is proposed to estimate the likelihood of each sensor reading for a known obstacle. The concept behind this approach is to model $P(z_t|S_t)$. This is sometimes referred to as a forward sensor model [15]. In [15], the authors suggest using maximum likelihood to identify the parameters of a multi-modal mixture distribution to model the likelihood. However, we attempt to directly model $P(z_t|S_t)$ using a convolutional neural network. The author of [13] pioneered the technique of learning a sensor model to predict an occupancy using two multi-layer perceptrons. We use the BEV scans of each $PC_t$ and predict 24 likelihoods, where each likelihood is the probability of the distance reading originating from a known obstacle. The $\{x,y\}$ are generated using BEV scans ($X$) and object ids that are emitted by the simulation instance ($Y$). Any known distance reading is assigned a probability of 1 and unknown 0. The network structure is shown and explained below:
The network takes in an input of $21 \times 21$ BEV scan and outputs a $24 \times 1$ vector where each element is a probability. The output units in the network use a sigmoid activation function. The network is trained using Stochastic Gradient Descent [34]. The convolutional layers use a ReLu [35] activations. The output of the network is used as a boolean mask where probabilities that are close to 1 are rounded to 1 and the rest are zeroed out. The mask is then inverted (using a logical not operation) and multiplied to $PC_t$ by broadcasting and elementwise multiplication. This can be formalized as:

$$ PC_t^* \leftarrow PC_t \otimes \hat{M} $$

(3.3)

Where $\hat{M}$ is the network output that is converted to a mask. The resulting point cloud $PC_t^*$ is then used to estimate the position of the human operator as discussed in further sections.

### 3.4 Estimation

In this section techniques and methods used to estimate the operator position are discussed.

#### 3.4.1 Centroid Estimation

The filtered point cloud $PC_t^*$ consists of points that probably unknown obstacles and can be used to estimate the operator ground position. To find the position, the centroid of $PC_t^*$
is found by:

$$\text{World } \hat{P}_{\text{Human}} = \frac{1}{n} \sum_{n=1}^{\text{size}(PC_t^*)} PC_t^*$$

(3.4)

$\text{World } \hat{P}_{\text{Human}} \in \mathbb{R}^2$ as the ground position can be 2 dimensional. $\text{World } \hat{P}_{\text{Human}}$ is essentially a 3 dimensional mean of the point cloud $PC_t^*$ with its z-axis removed.

### 3.4.2 Multi-Layer Perceptron

The multilayer perceptron (MLP) network is used as an alternate approach to averaging as averaging can lead to a bias when the input point cloud is temporally stacked, the past points will cause the operator position $\text{World } \hat{P}_{\text{Human}}$ to be biased. To overcome this issue, an MLP network is trained on ground truths $Y$ formed by $\text{World } p_{\text{Human}}$ originating from the motion capture system. The network is trained on $X, Y$ pairs such that $X$ entails the joint states of the robot at time $t$ and the point cloud $PC_t^*$. The joint states are included in the training data as they act as a context to the network. The network loss function was considered as Root Mean Square Error as we are dealing with motion data and the network can directly use the supervised loss. The loss function is shown below.

$$L = \sqrt{\frac{1}{n} \sum_{n=1}^{N} (\hat{y}_n - \text{true } y_n)^2}$$

(3.5)

The network consists of a ReLu [35] and tanh activations in the hidden layers and the output is identity. The network is essentially regressing to estimate a pose, $\text{World } \hat{P}_{\text{Human}}$ in continuous space.
3.5 Tracking

This section explains methods that are used to predict and estimate $\hat{P}_{\text{Human}}$ using a motion model and when no sensor data is present.

3.5.1 Kalman Filter Tracking Algorithm

The kalman filter [23] algorithm essentially predicts the output with a motion model and uses the output of the MLP networks to update its prediction. For the design of the filter the state was modeled as:

$$
\mathbf{x}_t = \begin{bmatrix}
  x \\
  y \\
  \dot{x} \\
  \dot{y}
\end{bmatrix}
$$

(3.6)

The measurement $z$ is two dimensional and given by $z_x$ and $z_y$ where they are obtained from the network output. The state transition matrix $A$ is defined as:

$$
A = \begin{bmatrix}
  1 & 0 & \Delta t & 0 \\
  0 & 1 & 0 & \Delta t \\
  0 & 0 & 1 & 0 \\
  0 & 0 & 0 & 1
\end{bmatrix}; \quad
Q = \begin{bmatrix}
  \sigma_x^2 & 0 & \sigma_{xx} & 0 \\
  0 & \sigma_y^2 & 0 & \sigma_{yy} \\
  \sigma_{xx} & 0 & \sigma_x^2 & 0 \\
  0 & \sigma_{yy} & 0 & \sigma_y^2
\end{bmatrix}; \quad
H = \begin{bmatrix}
  1 & 0 & 0 & 0 \\
  0 & 1 & 0 & 0
\end{bmatrix}
$$

(3.7)

$\Delta t$ is the time step which is approximately 0.0625 seconds. For the covariance, it is assumed that $z_x$ and $z_y$ are independent of each other, therefore a diagonalized covariance matrix is initialized.
3.5.2 Particle Filter Tracking Algorithm

For the particle filter a constant motion model was used. The particle filter algorithm used was implemented in [24]. The particle algorithm is shown below:

**Procedure 3.2 Estimation and Tracking Algorithm**

**Data:** Input Z

**Result:** X_t

1. initialize particles
2. initialize mlp
3. initialize timestep

**while true do**

1. **if** Z ≠ NaN **then**
   1. output = mlp(input)
   2. prediction = predict(particles, timestep)
   3. correction = correct(particles, output)
   4. estimate = estimate(correction)

2. **else**
   1. prediction = predict(particles, timestep)

end

In the particle filter algorithm, during the prediction stage the particles are subjected the motion model A as shown in Eq 3.7 and subjected to gaussian noise. After the prediction is made the particles are then resampled by using importance resampling and then an estimate is computed which gives the estimated state of the system. Different number of particles were chosen for the algorithm ranging from 100 to 1000.
Experiments and Results

In this section the results are discussed. A brief description of the experiment setup is given.

4.1 Experiment Setup

*Figure 4.1* Pictures showing the experiment setup. The simulation instance is shown on the left and the real world setup is shown on the right. The workspace is occupied by a human operator while the robot is performing a task.

The experiment setup can be broken down into hardware and software setups.

4.1.1 Hardware

The hardware setup of the experiment mainly comprised of Universal Robots UR10 [25] and an industrial factory floor like environment with props. The room was installed with motion capture cameras [27] and the robot had sensor rings mounted on three of its links.
To capture the state of the system $S_t$, a pick and place routine was on UR10 and data from robot joint position encoders, time-of-flight sensing rings and motion capture was collected. The data was also passed through a simulation instance where the complete model of the environment was already built. Along with collection of data from the physical system, data also captured from the simulation instance.

### 4.1.2 Software

The data was stored in rosbag files using ROS [26]. Rosbag allows the user to record and playback data and visualize at the same time which makes it extremely easy to debug. At the same time V-REP was used for running simulation instances and building the simulation-in-loop system. To enable communication between the physical system and the simulation system, a newly developed library for robotics research was used; PyRep [36]. PyReP exposed an interface to V-REP and allowed fast and efficient access which is very difficult, otherwise. To enable, inter-program communication, ZeroMQ [37] was used.

For training neural network models, TensorFlow 2.0 was used in conjunction with libraries such as numpy [38] and an OpenGL based plotting library called PyQtGraph [39].

To simulate the Universal Robots UR10 [25], UR Polyscope was used. A newly introduced protocol by Universal Robots called RTDE (Real-Time Data Exchange) was used to transport data in and out of the robot’s software and hardware. A diagram is given below to explain the communication architecture:
Figure 4.2 An illustration showing the hardware and the software components in a block diagram fashion.
4.2 Sensor Characterization

In this section, a single unit of VL53L1X is tested for noise and robustness under indoor conditions in white light. The picture of testing setup is shown below:

Figure 4.3 A picture showing the experiment setup for a single unit VL53L1X sensor. The wooden plank is the obstacle and the sensor is encircled in red. The distance between the obstacle and sensor is 600mm.

This test was vital to analyze the noise in the distance readings of the sensor. Building a noise model was also important as that allowed us to inject that noise in to the simulation instance to recreate a more realistic simulation. An obstacle with a width much less than the average depth of the human body to create strenuous conditions. The color of the object was kept comparable to human skin as the color of the object can have an effect on the amount reflectance of photos during time of flight sensing. Approximately 10,000 samples were collected over a distance of 600 mm under static conditions. After processing the noise data, the following plots were obtained.
Figure 4.4 The plot shows an error graph for the VL53L1X for a baseline of 600 mm.

Figure 4.5 The plot shows error distribution for the VL53L1X for a baseline of 600 mm.

It can be seen from figure 4.5, that the error is gaussian with a standard deviation of approximately 3.7 millimeters. To analyse the effect of obstacle moving away from the
optical center, the following plots were obtained:

**Figure 4.6** The plot shows an error graph for the VL53L1X for a baseline of 600 mm with the obstacle translated by 1 inch to the right side of the optical axis.

**Figure 4.7** The plot shows error distribution for the VL53L1X for a baseline of 600 mm with the obstacle translated by 1 inch to the right side of the optical axis.
**Figure 4.8** The plot shows an error graph for the VL53L1X for a baseline of 600 mm with the obstacle translated by 2 inches to the right side of the optical axis.

![Error Plot](image)

**Figure 4.9** The plot shows error distribution for the VL53L1X for a baseline of 600 mm with the obstacle translated by 2 inches to the right side of the optical axis.

![Error Distribution for Object close to the Optical Axis](image)
**Figure 4.10** The plot shows an error graph for the VL53L1X for a baseline of 600 mm with the obstacle translated by 1 inches to the left side of the optical axis.

**Figure 4.11** The plot shows error distribution for the VL53L1X for a baseline of 600 mm with the obstacle translated by 1 inches to the left side of the optical axis.
**Figure 4.12** The plot shows an error graph for the VL53L1X for a baseline of 600 mm with the obstacle translated by 2 inches to the left side of the optical axis.

![Error Plot](image1)

**Figure 4.13** The plot shows error distribution for the VL53L1X for a baseline of 600 mm with the obstacle translated by 2 inches to the left side of the optical axis.

![Error Distribution for Object close to the Optical Axis](image2)

All the results shown were obtained in conditions such that the obstacle was completely
inside the sensor field of view of 27 degrees. However, it can be seen from Figures 4.9 and 4.13, that as the object moves further from the optical axis the reliability of the sensor readings decreases due to the drastic increase in standard deviation values from 3.69 to 65.66 (2 inches to the right) and 80.76 (2 inches to the left).

After obtaining these plots, a distribution was modeled and used for creating artificial noise in the simulation. It was also noted that the manufacturer [17] also provides a corresponding standard deviation reading for distance observation. After analyzing the standard deviation readings. However, only the distances were used in this work.
4.3 Sensing Coverage Analysis

This section analyzes how much information do then time-of-flight sensors produce under controlled conditions that have been designed to mimic factory conditions. A robot and human are performing tasks in shared workspace. Their tasks can be broken down in to a trajectory of poses such the task trajectory of the human would be $\tau_{\text{human}}$ and the robot task trajectory would be $\tau_{\text{robot}}$. A plot of the $\tau_{\text{human}}$ has been shown further.

**Figure 4.14** The plot the trajectory of the human while performing an arbitrary task while working with the robot. The robot is located at the origin of the graph at (0,0).
It can be seen from Figure 4.14, that $\tau_{human}$, has an arbitrary structure associated with it on inspecting qualitatively. To analyze the coverage of the sensing system, distance between the robot and the human against the number of detections performed by the sensors has been shown. The human trajectory data was recorded in the real world setup and played back in the simulation and the robot performed the same task as $\tau_{robot}$, this allowed us to only capture the detections associated with the human in ideal conditions.

**Figure 4.15** The plot of distance versus number of detections performed by the sensors.

It can be seen form Figure 4.15 that as the proximity between the human and the robot reduces the number of detections increases. However, the max number of detections is only noted to be 5 which only occured once.
Figure 4.16 A bar graph showing number of occurrence of number of points observed per point cloud when the human is occupying the robot workspace.

On inspecting figure 4.16, it can be seen that the number of occurrences of point cloud with 0 points makes up for approximately 35.5% of the total distribution of the total point clouds observed in the experiment. Whereas, point clouds with a single point occur approximately 33% and a point cloud with 5 points is observed only once.
4.4 Data Association Experiments

4.4.1 Stochastic Masks Performance

The self detection convolutional neural network was trained on simulation data and then tested on two different dataset. One was synthetic whereas the other was recorded during the execution of $\tau_{robot}$ and $\tau_{human}$ in the real world. After the unfiltered point cloud data was obtained the test set were then passed through the network to obtain a filtered result. The output of the network was converted to a binary mask and multiplied by the input pint cloud. Please note that the point clouds were first converted to birds eye view images of resolution 0.2 meters.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class 0</td>
<td>0.77</td>
<td>0.89</td>
<td>0.83</td>
<td>18133</td>
</tr>
<tr>
<td>Class 1</td>
<td>0.34</td>
<td>0.17</td>
<td>0.23</td>
<td>5891</td>
</tr>
<tr>
<td>Accuracy</td>
<td></td>
<td></td>
<td>0.72</td>
<td>24024</td>
</tr>
<tr>
<td>Macro Average</td>
<td>0.56</td>
<td>0.53</td>
<td>0.53</td>
<td>24024</td>
</tr>
<tr>
<td>Weighted Average</td>
<td>0.66</td>
<td>0.72</td>
<td>0.68</td>
<td>24024</td>
</tr>
</tbody>
</table>

Table 4.1 Network Performance on the real world dataset during $\tau_{robot}$ and $\tau_{human}$.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>16156</td>
<td>1977</td>
</tr>
<tr>
<td>1</td>
<td>4865</td>
<td>1026</td>
</tr>
</tbody>
</table>

Table 4.2 Confusion Matrix for the Network Performance on the Real World Dataset.

The mean per class accuracy of the network over the entire test set was 53.25 % and occasionally peaked up to 70 %. We further show the results on the synthetic dataset which
was generated using the simulation environment.

<table>
<thead>
<tr>
<th></th>
<th>Precision</th>
<th>Recall</th>
<th>F1-Score</th>
<th>Support</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Class 0</strong></td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>21144</td>
</tr>
<tr>
<td><strong>Class 1</strong></td>
<td>0.99</td>
<td>0.99</td>
<td>0.99</td>
<td>2880</td>
</tr>
<tr>
<td><strong>Accuracy</strong></td>
<td></td>
<td></td>
<td>1.00</td>
<td>24024</td>
</tr>
<tr>
<td><strong>Macro Average</strong></td>
<td>0.99</td>
<td>0.99</td>
<td>1.00</td>
<td>24024</td>
</tr>
<tr>
<td><strong>Weighted Average</strong></td>
<td>0.99</td>
<td>0.99</td>
<td>1.00</td>
<td>24024</td>
</tr>
</tbody>
</table>

The mean per class accuracy observed to be **99.67 %**.

The output of the self classification is shown further.

**Figure 4.17** A point cloud shown on the left with its corresponding bird’s eye view on the right. The $z$ values are mapped to rgb values and are shown as a colormap. These obtained by applying the output of the network to the input pointcloud.

![Point Cloud](image1.png) ![Bird’s Eye View](image2.png)
Figure 4.18 This plot shows the network output overlayed on top of the human trajectory.
**Figure 4.19** This plot shows the ground truth for the network overlayed on top of the human trajectory. The ground truths are generated from a physics engine instance.

**Figure 4.20** A point cloud shown on the left with its corresponding bird’s eye view on the right. The $z$ values are mapped to rgb values and are shown as a colormap. These obtained by applying the output of the network to the input pointcloud.
Figure 4.21 This plot shows the network output overlayed on top of the human trajectory. These results show the inference on the synthetic dataset.
Figure 4.22 This plot shows the ground truth for the network overlayed on top of the human trajectory. The ground truths are generated from the physics engine instance.

It can be seen in Figure 4.22 that the points in the point cloud align with the human trajectory.
4.5 Estimation and Tracking Experiments

For testing the estimation and tracking, random walks were generated in a simulation environment for training data and were tested on a predefined trajectory. Therefore, to train the estimation and tracking for the MLP, a 2 dimensional uniform distribution was sampled for generating synthetic human positions.

Figure 4.23 This plot shows a 2D density histogram of the training data used for the human position. The hollow region in the middle is the part of the volume that cannot be accessed by the human operator as it is occupied by the robot and other entities.
4.5.1 Centroid Performance

Figure 4.24 This plot shows the predefined ground truth trajectory for the human operator in a simulation overlayed by the centroid of the points observed by the sensor.

It can be seen in Figure 4.24 that the centroids are only obtained sporadically and there is no sensing information available during that time.
4.5.2 Kalman Filter Performance with Centroid Estimation

Figure 4.25 This plot shows the predefined ground truth trajectory for the human operator in a simulation overlayed by the kalman filter output.

It can be seen in Figure 4.25 that the centroids are tracked using a linear kalman filtering algorithm and diverges frequently.
4.5.3 Particle Filter Performance with Centroid Estimation

**Figure 4.26** This plot shows the predefined ground truth trajectory for the human operator in a simulation overlayed by the particle filter output.

It can be seen in Figure 4.26 that the centroids are tracked using a particle filtering algorithm.
4.5.4 MLP Performance

**Figure 4.27** This plot shows the predefined ground truth trajectory for the human operator in a simulation overlayed by the MLP output of the points observed by the sensor.

It can be seen in Figure 4.27 that the centroids are only obtained sporadically and there is no sensing information available during that time.
4.5.5 Kalman Filter Performance with MLP Estimator

Figure 4.28 This plot shows the predefined ground truth trajectory for the human operator in a simulation overlayed by the kalman filter output.

It can be seen in Figure 4.28 that the MLP outputs are tracked using a linear kalman filtering algorithm and diverges frequently.
4.5.6 Particle Filter Performance with MLP Estimator

Figure 4.29 This plot shows the predefined ground truth trajectory for the human operator in a simulation overlayed by the particle filter output.

It can be seen in Figure 4.29 that the MLP outputs are tracked using a particle filtering algorithm.
**Table 4.5** This table shows the RMSE values for the plots shown before.

<table>
<thead>
<tr>
<th></th>
<th>MLP (RMSE)</th>
<th>Centroid (RMSE)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Raw</strong></td>
<td>0.8890</td>
<td>0.9011</td>
</tr>
<tr>
<td><strong>Kalman Filter</strong></td>
<td>1.2304</td>
<td><strong>0.9032</strong></td>
</tr>
<tr>
<td><strong>Particle Filter</strong></td>
<td>0.8705</td>
<td>0.8668</td>
</tr>
</tbody>
</table>
Discussion and Future Work

5.1 On Sensor Characterization

On analyzing the sensor data obtained in section 4.1, it was observed that the laterally translating obstacle with respect to the optical axis of a single unit of the VL53L1X module [17] whilst within its field of view (FoV) of 27 degrees caused the sensor reading to be more unreliable as the obstacle approached near the edge of the FoV. Also, as the sensor only provides a single distance reading, the confidence of that is greatly affected by a lot of factors such as the speed of the motion of the sensor while on the robot, specular reflections, obstacle shape and size and so on. Also, since eight sensors were used per ring, a need for a greater number of sensors is evoked as overlapping FoVs can be used to drastically reduce the uncertainty in observing a 3D point.

As shown in section 4.4, it can be seen that the rudimentary ray casting based sensor model does not provide an output good enough to match the model shown in preceding sections. Therefore, a more sophisticated sensor model is needed to learn the environmental representation.
5.2 On Sensing Coverage Analysis

The key takeaways from section 4.3 include the fact that task trajectories can be extremely well structured and can be leveraged for other higher level perception and planning techniques such as task estimation. In figure 4.14, a clear structure can be observed, and it can be seen that the human operator stopped in approximately 3-4 different regions to perform a step in the task. Also, one of the areas towards the top part in the plot is out of range of the sensor system.

On observing Figure 4.15, it can be seen that the number of detections is inversely proportional to the distance between the operator and the robot which implies that this sensing system can perform better when the human and robot are working in very close proximity to each other. Combining this sensor system with a setup like a 2D laser range finder would open new avenues in close proximity tracking while maintaining robustness.

On inspecting Figure 4.16, a large proportion (35.5 %) of the observed point cloud samples consisted of zero points. Since, the plot was obtained when the human was occupying the workspace. It can be safely said that the sensing system does cannot observe the workspace completely. Therefore, the setup has blind spots.

5.3 On Data Association

It can be seen in tables 4.1 and 4.3 that the classifier performance greatly varies. This could be due to change in the domain in the data which implies that the real world data is afflicted by much more complex noise which cannot be modelled completely which lead to a poorer performance. However, as the classifier was solely trained on simulation data,
the performance on real world data shows positive signs. Pretraining on simulation data and fine tuning on real world data must be explored. Also, leveraging more temporal information should be explored.

5.4 On Estimation and Tracking

On observing Figure 4.19, we can see that the sensing data is not present at all times even when the human is occupying the robot workspace. The sensor setup does not produce any new information for extended durations. This can be reasoned by an intuition that the observability of the sensors is also dependent on the robot and human task trajectories. Another key takeaway is that combining this setup with other sensing modalities can be used to build towards a robust perception pipeline. It can be seen in Table 4.5, that particle filter showed the least RMSE error, however, the Kalman Filter tracker diverged several times and the performance worsened as shown by the error value obtained in Table 4.5. The particle filter performs well because of the non-linear nature of the trajectory.

On visually inspecting Figures 4.24 and 4.27, it can be seen that the MLP Estimator output is less biased as compared to the centroid output. The weights of the network are scaling the input points whereas the biases are translating them, thereby, forcing the output to move towards the true trajectory. The centroid estimation on the other hand lead to a biased estimation as the points observed lie on the outer surface of the human operator’s body, therefore, the output is biased by the position of those points.

On analysing Figure 4.28, we can see that, the Kalman Filter Performs poorly which was due to the very high variance of the MLP estimator output. The filter diverged repeatedly leading to a high very a RMSE error.
5.5 Future Work

As the MLP and Centroid estimation provide two different perspectives. Obtaining a weighted average of the two could yield better results. Also, as the observations obtained by the sensing setup are extremely sparse in nature, having pre-defined task knowledge can enable detection and tracking of the human operator. A scheme to perform local trajectory matching against a global task trajectory could be used to determine the human operator’s presence.

In [6], the sensing setup using 16 units per ring were used and the system yielded better results. Therefore, using more sensors per ring to boost the coverage will definitely improve the perception capability of the system.

Also, since this setup essentially provides only the partial pose information. This information can be leveraged in a sensor fusion scheme where an inertial measurement device can be mounted on the human operator’s vest. Fusing the two sensing modalities can allow the system observe the operator’s position for extended duration(s).

As the self detection convolutional neural network performed badly on the real world data as compared to the synthetic data performance. It could be reasoned due to the inherent noise in the input data. Therefore, a transfer learning approach or a fine-tuning approach the weights on a small real world dataset should be adapted.
Bibliography


