

An Approach to Detection, Extraction and Localisation of Train Features using RGB-D Data

Julian Malki

Student Number: 11058637

Project Number: A19-005

Major: Mechanical & Mechatronic Engineering


Supervisor: Alen Alempijevic

University Of Technology, Sydney

1. Statement of Originality

Declaration of originality: The work conducted in this, other than that specifically attributed to another source, is that of the author(s) and has not been previously submitted for assessment. I understand, should this declaration be found to be false, disciplinary action could be taken and penalties imposed in accordance with University policy and rules. In the statement below, I have indicated the extent to which I have collaborated with others, whom I have named.

Signature of student:



Date: 28/10/2019

2. Abstract

It is without question that the use of computer vision systems has greatly aided the growth of automation in the transportation industry, the future of the industry is increasingly reliant on advances in the field. There are various technical challenges including imaging in various environmental conditions, distributed sensing networks, processing data at high speeds among others. In such systems the need for image classification in real time for such applications is expected and there has been extensive studies and development of various methods of approach. An approach becoming increasingly common is the use of Machine Learning (ML) and the subset Deep Learning (DL) algorithms in the use of object detection.

This project is based on an approach to handling object detection in a real time application as part of an autonomous public transportation management tool developed in conjunction with UTS CAS and Downer Rail. In this project we will take a look at the methodologies that have been used in similar applications and evaluate a design framework that performs best with the constraints of the project. We will then provide a general implementation and provide our recommendations for further research and development in the project.

3. Contents:

1. Statement of Originality
2. Abstract
3. Contents
4. List of Figures/Tables
5. Introduction
6. Literature Review
7. Design Process
8. Implementation
9. Evaluation
10. Conclusion
11. References

4.

4.1 List of Figures

Figure 1: Theoretical Framework flowchart

Figure 2: Primesense RGB-D Camera

Figure 3: Camera Locations

Figure 4: Original Depth Image

Figure 5: Point Cloud Converted From Depth image

Figure 6: Point Cloud with Detected Plane in Red

Figure 7: Point Cloud with Transformed Plane in Green

Figure 8: Depth Image Converted From Remaining Point Cloud

Figure 9: Labelled Depth Image

4.2 List of Tables

Table 1: Design Criteria

Table 2: Evaluation of Design Criteria

5. Introduction

This paper presents an approach to solving a key task identified in the development of the DwellTrack Project, a partnership between Downer Rail and UTS CAS. The current system uses RGB-D data to track key parts of a public train station, by applying image processing and machine learning fundamentals, subnets of the systems are being developed to support typical computer vision tasks. More specifically, in this paper we address the task of feature extraction, identification and tracking of key train features such as train doors and windows.

There is a wealth of literature available on the developments in Computer Vision (CV) and Machine Learning (ML). Below we take a look at some of the research conducted in the field that can assist us in our application and make design choices based on the scope of our project.

The constraints of the project should play a key role in determining our design and implementation method. Unlike other applications we consider the use of depth images only, this format presents several challenges in the task. Given the anticipated use case, it is necessary that whatever method we choose to use we can achieve our results in a real time frame rate.

In the Design section we will outline our theoretical framework, our design criteria and the project scope. A detailed breakdown of how the system was implemented is also included. This includes specific information on the initial data format, 3D image processing, using the data to train an object detector deep learning model and publishing output data. Following our implementation we will assess the results of the designed system and make our evaluations for future improvements.

5.1 Research Question

This project aims to address the following question:

“How can we effectively use a combination of traditional image and 3D processing techniques with deep learning models to detect significant train features in real time using single channel depth images?”

This question allows us to consider a variety of tools, frameworks and methodologies available to solve our task. It also addresses the specific constraints of the project; data format and the processing rate required.

With a clear definition for our task we can selectively research further to narrow down our options and test our initial design choices with basic feedback. After we complete implementation of our proposed system we are able to identify drawbacks and evaluate the solution with respect to the original research question.

6. Literature Review

In this section we take a look at some of the research conducted during the design process and the impact previous work conducted had on evaluating our design methodology.

Several publications have been made in relation to the work conducted by Alen Alempijevic and UTS CAS, here we take a look at the some of the major developments in the project thus far.

In 2015 “Foundation technology for developing an autonomous Complex Dwell-time Diagnostics (CDD) Tool” was published by our project supervisor Alen Alempijevic (et al.). The paper conducts research into using computer vision techniques to autonomously manage data collection, analysis and directives for train platform operations in order to reduce ‘dwell’ time between the trains arrival and departure. Dwell time is defined as the time taken during the trains arrival, passenger alighting/boarding and the train departure. Again the use of depth images was mainly focused on given the nature of privacy concerns. Traditional feature detection methods are used to extract key feature points that make up features of the train and the train platform. These key features are used to extract the key geometrical features of view ie platform orientation, platform limit, train windows and train carriage size. The feature detection was also used to indicate key events in the dwell time such as train arrival, train door opening, passenger alighting/boarding, train door closing and train departure. The results delivered were graphed showing the dwell events detected using the system and the dwell events detected manually. Overall the accuracy was quite impressive however it is notable that the algorithm struggled to detect differences in Door Opened and Departure events.

Another key component of the project is the ability to manage passenger movements throughout the station. “A robust people detection, tracking, and counting system” Alempijevic A(et al.) proposes an approach to extracting the passenger movement and behaviours, again the project is limited with privacy concerns over using facial parts of the RGB images therefore the research is conducted using depth images only. Most human tracking algorithms used some facial features to track people through the image however in this case due to the limitations a Head Shoulder Signature (HSS) is developed. The HSS is used to track each person through image while determining there orientation and furthermore there intention in their passenger movement. A Support Vector Machine is implemented to automate the detection of the HSS given a pre defined feature model. Using the HSS, business information available (train departure times) and the typical choices for a passenger depending on their position within the station the system takes an approach to monitoring passenger behaviour.

Image classification in real time using computer vision techniques vs deep learning

Machine learning is a relatively new technique used in image classification, in order to evaluate the design of our system we must also consider the application of traditional computer vision object detection methods. Typically in the past we have seen feature detection used to customise object detection for very specific applications. Feature descriptors such as SIFT, SURF and BRIEF can be used along with typical CV algorithms such as edge detection, corner detection, hough transforms, threshold segmentation and so forth. 'Real-Time Object Detection Using Distance Transforms' Gavrilu, D.M. (et al.) paper also discusses the use of using distance transforms in a real time image processing application, the algorithm uses templated shapes between frames to select 'on/off' assignment for each pixel by matching against the template shapes, a hierarchy or probabilistic value can also be applied to the template shapes. Generally using these techniques the user will need to manually determine objects and the corresponding feature matches to model and detect that object in future instances. In previous cases we have also seen the use of customised Support Vector Machines (SVM) used to classify images based on a feature model that was previously extracted, this can be considered as a step toward assisted machine learning in image classification.

More and more recently machine learning has been used to assist with image classification. A typical model uses an artificial neural network (ANN) to facilitate learning by using a credit assignment to regions of interest (ROI) through multiple layers, the output is a probability that the object detected matches the pre trained classes (objects that were annotated in the images). Deep learning is derived from a typical machine learning model and gets its name 'deep' due to the extension of multiple pairs of convolutional and pooling layers. In order to recognise a ROI the model is trained with clearly annotated images at this stage weights may be used to identify low level edge detection or in high layers larger general shapes closer to the actual trained class.

Comparing traditional CV to deep learning models we can see there are a lot of immediate advantages. We can see rapid improvements in computing power, memory capacity, image sensor resolution required and the general cost-effectiveness of the system. DL allows engineers to achieve greater accuracy with a less expert analysis. However training these systems can be quite time consuming and computationally expensive therefore given the application it may not make sense to always use DL opposed to traditional CV for image classification. As referenced by "Deep Learning vs. Traditional Computer Vision" O' Mahony, N. (et al.) typically there are some tasks which DL models are not well suited for yet such as images with multiple instances of multiple objects generally don't perform well, visual SLAM and 3D vision are also not typically applicable. Although the development of 3DCNNs are progressing however typical DL models are not accustomed to handling 3D convolution structures and typically we do not see DL models trained with 3D structures such as point clouds.

Ideally we should use a combination of traditional CV and DL to assist with our task. Image enhancements, rotations, translations, scaling and other image processing techniques can

greatly improve the accuracy of the DL model and reduce the computational load and timing of training the model.

YOLOv3 vs Mask R-CNN

As part of the design process it is key that we consider the various Deep Learning models available for use. Generally there isn't a straight answer to the best model as there is a constant tradeoff between accuracy and speed. The image data format and the system tools we have available also apply a large role in our decision making. For this section we took a closer look at YOLOv3 which is one of the more recently highly proposed methods and Mask R-CNN which provides image classification and instance segmentation.

Mask R-CNN (He et al., 2017) extends from Faster R-CNN (Girshick 2015) to provide an object mask in parallel to the existing classification and localization branch. R-CNN methods generally use a sliding window approach that determines bounding box regressions between each pooling layer checking sections at a time. The method You Only Look Once (YOLO) (Redman 2015) divides the entire image into a grid structure and assigns a confidence score that the grid contains a part of the pre trained class, allowing classification and bounding box regression to happen at the same time.

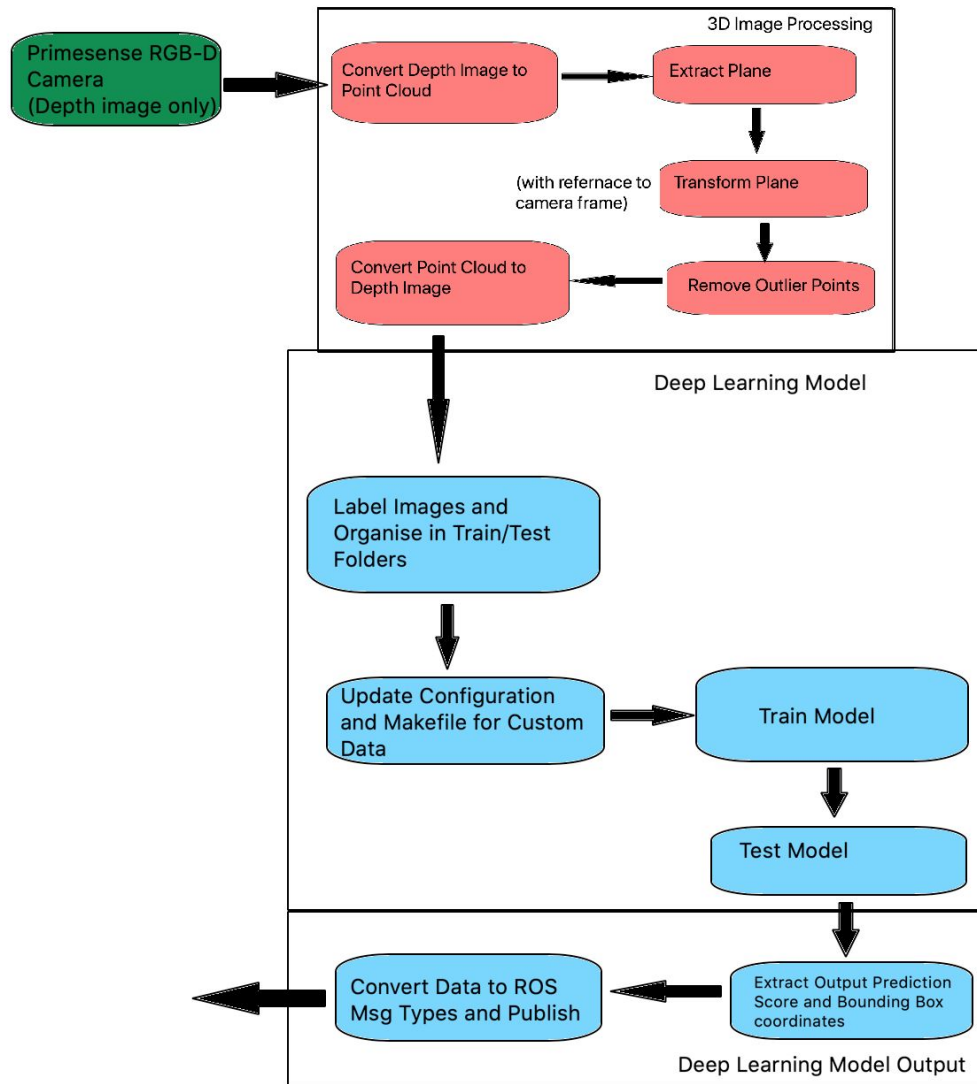
The extensive research available proves the major differences and advantages between the two models to be:

- Mask R-CNN has a greater accuracy if speed is not a concern and performs well even in images with multiple objects clustered in the images
- YOLOv3 performs great on a live video frame rate opposed to Mask R-CNN applications which generally run at a max rate of around 10Hz
- YOLOv3 may struggle with smaller objects in images or multiple instances of objects that are close to each other
- YOLOv3 sees the complete image at once providing some contextual information that helps with avoiding false positives

7. Design Process

Theoretical Framework

Figure 1: Theoretical Framework Flowchart



Design Criteria

Table 1: Design Criteria

No.	Criterium	Justification
1	Proposed design should be based on insights gained during research and testing	In order to provide a well suited solution we need to explore the work of other proposed solution for which there are many. We need to assess the

		common attributes of other systems and translate that knowledge to customize our system to best suit the needs of the project scope.
2	Proposed system should outperform other image classification with traditional CV methods	In order to evaluate our proposed solution we need to compare our work to other systems in similar applications. To be considered successful the project should outperform other methods with considerations to the differences in hardware.
3	The model should predict the train doors within an accuracy tolerance of +/- 10%	The accuracy of the model predictions is a key metric of how the model performs. We benchmark a tolerance of 10% as a reasonable deviation for our mean Average Precision (mAP) values. We are also able to retrieve our loss function values that denote the differences between prediction scores and ground truth objects during training.
4	The model should provide a prediction with 100 ms	The model should be able to be executed in 100ms (10Hz) or faster. This allows the prediction to be executed in real time on incoming images.
5	Code should be reusable and modular within confines of the application (different camera locations, different camera types etc)	As the application is part of a larger system it is ideal that code can be expanded, reusable and modular in its design. Well documented code will also ensure following users can expand on its development.
6	The proposed system should consider time synchronisation, buffering and validation of data	Given the application should be used as part of a larger system we need to consider the transfer of data across multiple nodes.
7	System should be setup with minimal startup cost	The proposed system should be able to implemented with few changes. Should not require hardware that is currently not used within the project.
8	System should output data in a format that is consistent with input data format	System should output data in the form of ROS message types

Scope of the project

The scope of this project is to deliver a package of code that includes a ROS node and custom trained model of the YOLOv3 deep learning framework. The tools delivered should allow users to setup the implementation and training of the model during the initial setup of the camera location. More specifically the ROS node includes an executable that will receive depth image messages via a ROS 'subscriber' and post processed depth image messages via a ROS 'publisher'. The ROS format of this setup is key as the code delivered should be modular and reusable with the current system tools. Ideally the system will not require storing any large datasets as these largely consist of raw uncompressed images of RGB and depth images received at a rate of 30Hz which can quickly equate to significantly large files. The proposed solution is a system that can be setup with little customisation. With minor setup work the system should transform the images, train the DL model and later use the trained DL model for automated feature detection of the train features.

It is important to note that this system has been developed for a specific application although some parts may be reusable outside of the scope of this project with minor adjustments.

Some aspects which fall outside of the scope of this project are:

- Trained model functionality with RGB images - for this project the given data consisted of raw depth images as such all image processing techniques and the deep learning model were implemented to use 16 bit, single channel image types.
- Image plane extraction and transforms - the code responsible for extracting the train door plane and transforming the points within that plane assumes that depth images are derived from a camera that is down angled and facing the train from an elevated position to ensure the train doors plane in the largest plane in the field of view.
- Model training with GPU - we are training the model using a Nvidia GTX745 graphics card. The hardware available in the end use case is still unclear and as such it is recommended that the training happens remotely once enough data has been collected in the initial stages. Once the model has been trained it can be used in a real time application with very limited hardware resources. YOLOv3 claims the ability to process images at a rate of 45 frames per second and can be trained using only a CPU, although it is recommended to use a GPU with at least 4GB in memory.
- Using implementation without ROS - we have used ROS to transfer the depth images, in order to be used without ROS the user will need to update the passing variables and their data structures.

8. Implementation

System overview

- Depth camera publishes images to topic
- Preprocessing subscribes to image topic and publishes to image files
- Image files are annotated and sorted into respective train and test folder
- DNN model trained using train image set
- DNN model tested using test image set
- Model loaded into application hardware and run with real time frame rate
- Output data is published to topic

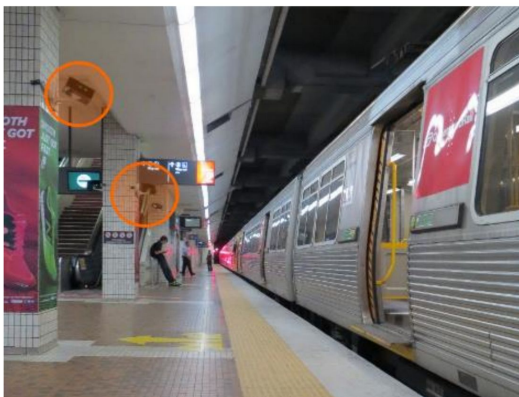
Data format

The data is collected via Primesense RGBD cameras located on the train platforms. The data available was collected in short samples in ROS bag files and is encoded as 8 bit single channel raw depth images. We convert the format to 16 bit CV_32FC1 in order to uncap the depth values to a maximum value of 65535 (opposed to 255 limit with 8 bit images).

Figure 2: Primesense RGB-D Camera



Figure 3: Camera Locations



Pre processing ROS node

This section of code was developed as a ROS node that can be used either during training or testing (with minor adjustments). The design choices made in this stage were made with a focus to reduce the overall training costs at first setup of the model. We understand from our research that reducing the image and unnecessary parts of the image will allow training to complete faster and should not include an accuracy tradeoff. As depth images have noisy edges we are unable to use traditional CV feature detection methods to extract key parts therefore we mainly make use of PCL to assist us with our 3D vision image processing.

The different stage of our preprocessing node are:

Subscribe to the `/depth/image_raw` topic

- Here we use the ROS subscriber format to callback image data at 30 frames per second, the data is converted from a ROS message to OpenCV type image with 16 bit single channel image structure

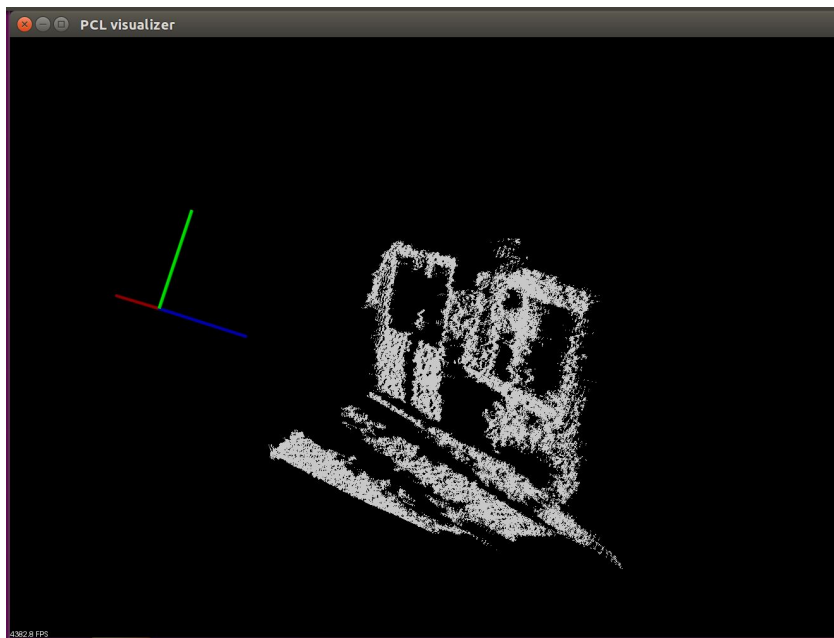
Figure 4: Original Depth Image



Convert the image structures into a PCL point cloud

- We convert the image X,Y,Z coordinates to PCL point parameters using the pixel image values for the X/Y values and the depth readings for the Z values. The data is converted to metres within reference to the camera frame as the origin
- After building the point cloud we downsample it using the voxel grid filter to reduce the computational load of handling a large point cloud

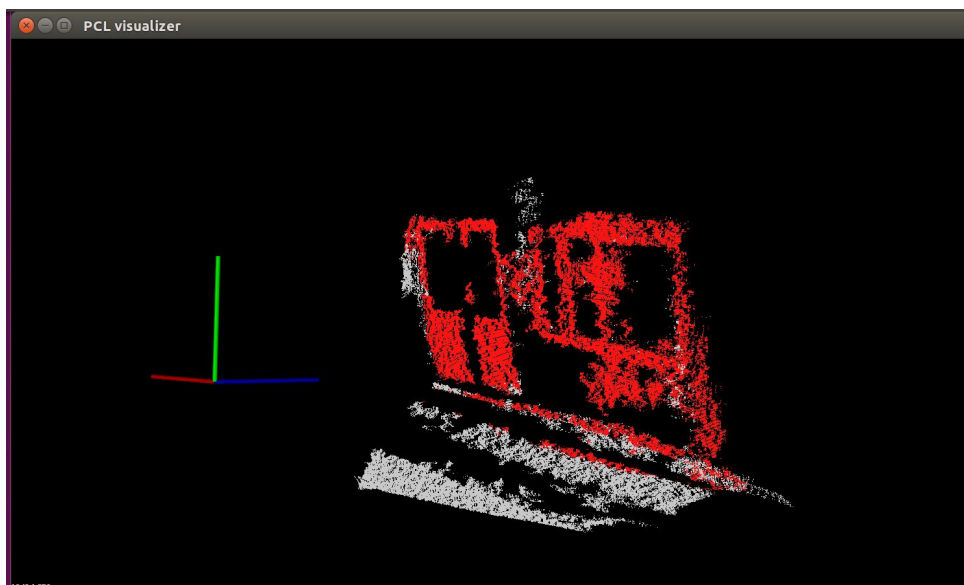
Figure 5: Point Cloud Converted From Depth image



Detect the plane that the train door lies in

- Using the PCL RANSAC method we search the cloud for the largest plane that has minimum point density threshold of 0.15
- Create a new point cloud with only the plane inliers

Figure 6: Point Cloud with Detected Plane in Red

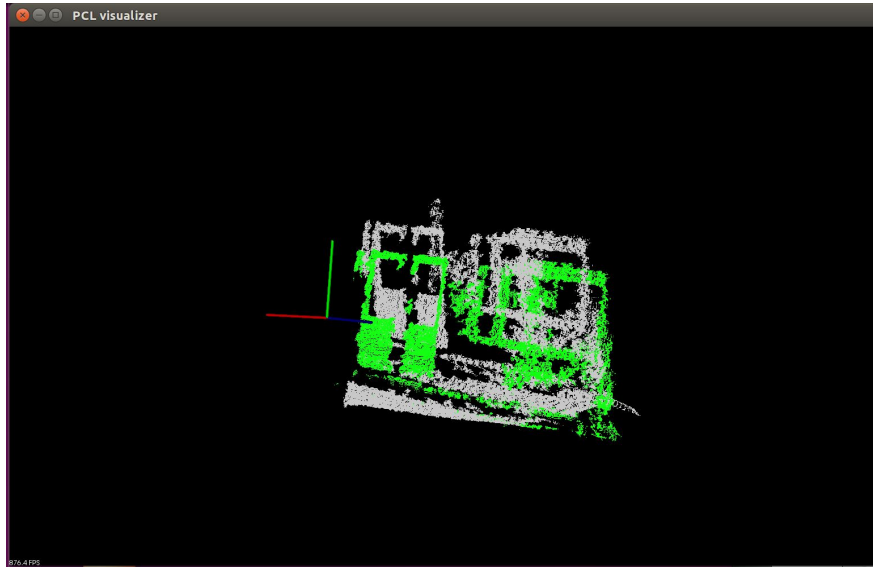


Compute the transform from extracted plane normal to origin (using the XY unit vector)

- We create a unit vector for the plane normal using values from the plane model coefficients
- Use the cross product to determine the rotation between the unit vectors
- Use the minimum and maximum X,Y point cloud values to determine the centroid of the point cloud and compute translation values for the transform

Transform inlier point cloud into XY plane (perpendicular view) using transform

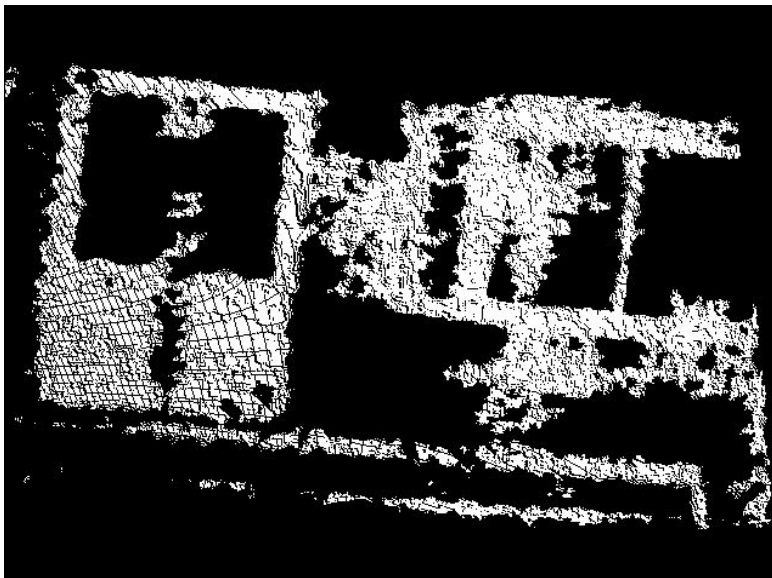
Figure 7: Point Cloud with Transformed Plane in Green



Convert remaining point cloud into depth image

- Create Mat object and assign x,y and depth values
- Resize image structure to reduce width and height
- Convert image structure to ROS type message

Figure 8: Depth Image Converted from Remaining Point Cloud



Publish the images via topic or write to image files

- For initial training the we will write the images to jpeg files
- For executing (testing) the DL model we will retrieve the images via the /plane_converted/image_raw topic

Labelling data and Virtual Environment Setup

Before implementing our model we need to setup our virtual environment to use the necessary python machine learning tools. Install the Keras and TensorFlow libraries in the virtual environment. We also need to create symbolic links to any previous installed version of python or OpenCV that we intend to use and add them to the path using the bash startup script.

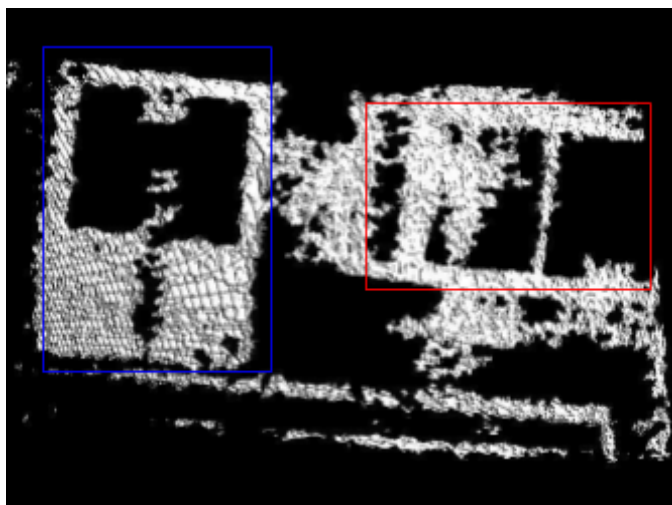
At this stage we need to prepare the image data into a format that the YOLOv3 model can analyse. After removing the unnecessary images that did not include our class we are left with around 520 images. We use AlexeyAB Yolo_mark (https://github.com/AlexeyAB/Yolo_mark) tool in order to manually outline boxes in 80% of the images (designated for training), testing images are not annotated. Once we mark the box we will also get a labels text folder that will consist of the following output:

```
<class_id><label_min_X><label_min_Y><label_max_X><label_max_Y>
```

Prepare the folders for the Train and Test split (80% / 20%) of the images each should have an Images folder and Labels folder containing images and annotated text files with respective name matching.

Update the configuration files to enable our GPU in this case we are using the Nvidia GTX745 with 4GB of memory, enable CUDNN we are using CUDNN 8.0 and enable OpenCV.

Figure 9: Labelled Depth Image



Training and testing the DNN model

Once everything is ready, we can train the model using our annotated images. Execute the Detector training script included in the Darknet repository ensure to update the correct paths to configurations files, Labels and Images folders.

This can take quite a while as we used quite a small memory GPU. In order to assist with our training time we've reduced the number of batch and subdivisions in the YOLOv3 configuration file. Once the code has completed the specified iterations we can analyse the output results of the train.log file.

If we are happy with the mean Average Precision (mAP) value and the loss value is near zero we can begin to use our model with test images. Update the configuration file to adjust batch and subdivisions for testing and run the Detector test script in Darknet with the test images.

Model output format

The output format is a text file with similar to the labels text file format except the data also includes a prediction score value. It is then suggested that the images and their corresponding results values are extracted and published to a ROS topic for extended use.

9. Evaluation

In order to evaluate the system we can assess how well the design criteria were met:

Table 2: Evaluation of Design Criteria

No.	Criterion	Evaluation
1	Proposed design should be based on insights gained during research and testing	We have implemented a solution based on the research conducted. The Literature Review section outlines some of papers that influenced our key decisions
2	Proposed system should outperform other image classification with traditional CV methods	The proposed model can outperform traditional CV methods by reduced computational loads, processing time and memory required
3	The model should predict the train doors within an accuracy tolerance of $\pm 10\%$	This metric has not yet been evaluated.
4	The model should provide a prediction with 100 ms	The original model boasts a frame rate of 45 FPS however depending on the system architecture it is likely we can expect around 20 FPS which satisfies our criteria
5	Code should be reusable and modular within confines of the application (different camera locations, different camera types etc)	The code provided requires little to no changes to reuse transformation methods.
6	The proposed system should consider time synchronisation, buffering and validation of data	Given system is implemented with ROS we allow time synchronisation to be handled by ROS spin. We increase our queue sizes and count messages to ensure data is not lost.
7	System should be setup with minimal startup cost	The proposed system can be setup on a system with at least a typical Intel i5 processor and requires a GPU that has a minimum memory of 4GB
8	System should output data in a format that is consistent with input data format	System output is in the format of ROS message types

In order to effectively evaluate our design we need to consider the strengths and weaknesses of our system. The ability to reuse the 'Pre Processing Node' across multiple camera locations is ideal meaning that the only time consuming part of setup is training the

DL model. However it is more effective to train the model remotely or on a master machine while using the camera hardware to remain publishing the raw images only.

We encountered various challenges with installation and setup of the various frameworks and libraries used throughout this project. PCL, CUDA and YOLOv3 all consist of large sets of files and can be quite time consuming to compile and build. It is recommended that users setup the necessary tools in a virtual environment and test the tools are available/enabled by using a simple python script.

10. Conclusion

In this paper we conducted research into the prior work of the DwellTrack project, past development into traditional CV image classification methods and open source deep learning object detector models. We proposed a solution to performing real time image classification on single channel depth images, assisted with machine learning. We explained the step by step implementation of the system with the hope that future users find value in our paper.

Given the rapid developments in machine learning we can expect the concept to become more commonly used with increasingly accurate results on lower computational costs. Given these advancements we would encourage future users to consider the benefits of any future released models. Given the large manual task of annotating images for these we anticipate the arrival of a cleaner solution to training these models either in the form of reduced image data set sizes required or a semi automated approach to labelling the images ie. detecting a region of key features that make up an object without identifying the object.

11. References

Alempijevic, A. (et al.) 2015, 'Foundation technology for developing an autonomous Complex Dwell-time Diagnostics (CDD) Tool', *Australian Transport Research Forum*

Alempijevic, A. (et al.) 2015, 'Sensing and perception technology to enable real time monitoring of passenger movement behaviours through congested rail stations', *Australian Transport Research Forum*

Alempijevic, A. (et al.) 2014, 'A robust people detection, tracking and counting system', *Australasian Conference on Robotics and Automation*

O' Mahony, N. (et al.) 2016, 'Deep Learning vs. Traditional Computer Vision', *IMaR Technology Gateway, Institute of Technology Tralee, Tralee, Ireland*

Gavrila, D.M. (et al.) 1999, 'Real-Time Object Detection Using Distance Transforms', *IEEE International Conference on Intelligent Vehicles, Stuttgart, Germany*

Chandan, G. (et al.) 2018, 'Real Time Object Detection and Tracking Using Deep Learning and OpenCV', *Australasian Conference on Robotics and Automation*

Redmon, J. (et al.) 2018, 'YOLOv3: An Incremental Improvement', *University of Washington*

He, K. (et al.) 2017, 'Mask R-CNN', *University of Washington*

Darknet github repository, viewed 10 October 2019, <<https://github.com/pjreddie/darknet>>

AlexeyAB - Yolo_mark, viewed 10 October 2019, <https://github.com/AlexeyAB/Yolo_mark>

