Leveraging Machine Learning for Collision Avoidance in Continuous-Wave Time-of-Flight Cameras

Yoganand Biradavolu
ybiradavolu@wisc.edu

Yuchen Gu
ygu48@wisc.edu

Saurabh Kulkarni
skulkarni27@wisc.edu

Sophie Stephenson
srstephenso2@wisc.edu

1 INTRODUCTION

Continuous-wave time-of-flight (C-ToF) cameras are growing increasingly popular for 3D imaging applications such as robotics [13], augmented reality [8], and autonomous vehicles [5]. Many of the usage scenarios for C-ToF cameras require accurate measurements at all times, particularly for scenarios like autonomous driving where incorrect measurements could lead to a dangerous crash. Unfortunately, C-ToF cameras suffer from multi-camera interference (MCI), making it difficult to achieve good accuracy when more than one camera is present.

Prior work has attempted to solve the problem of MCI. Some have proposed time-division multiplexing methods to avoid interference completely [18]; these methods require synchronization, which is unrealistic in real-world situations. Other prior work uses modulation techniques, such as providing each camera with a unique modulation frequency(e.g., [14, 19]). Unfortunately, these solutions are not scalable since there is a limited number of usable frequencies, and additionally any method which uses sinusoidal modulation requires hardware upgrades. Recent work from UW Madison proposed stochastic exposure coding (SEC), which fires C-ToF cameras with a certain probability during set time slots [11]. Though this method is more promising than the other solutions, its probabilistic nature does not completely avoid interference.

We are unaware of any 100% effective collision avoidance solutions that make full use of the main channel without the use of side-channels to explicitly communicate. While the most effective solution most likely involves additional hardware and explicit communication and synchronization, we wanted to explore the possibility of using implicit side-channel communication instead. Each collision scenario has a unique fingerprint, at least in the context of autonomous driving. This fingerprint is composed of the number of C-ToF cameras in the scenario, the velocity of each actor, obstacles, etc. There is some optimal distribution of the channel for each scenario that leads to the most information gain across all actors. For example, a car that is stopped at a red light does not need the channel as much as the car going through the intersection. Thus, if this stopped car were able to detect that it should not use the channel, the other cars in that situation would be able to use the channel more optimally. With this motivation, we take steps towards designing a machine learning black box that can identify when a C-ToF camera should fire. We envision a scenario where each C-ToF camera is equipped with a companion RGB camera which records videos of the scene. For each frame captured by the RGB camera, the black box tells the C-ToF camera whether it should fire. Ideally, this black box can identify multi-camera scenarios and allow the cameras to share the medium without explicit synchronization.

In reality, the presented problem is difficult to address in many aspects. Thus, in our project, we contribute prototypes and observations towards our ideal solution and raise challenges for future work, including alternative approaches.

UW Madison CS707, December 22, 2021, Madison, WI
2021. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

2 BACKGROUND & RELATED WORK

2.1 Continuous-Wave Time-of-Flight Cameras

Continuous-wave time-of-flight (C-ToF) cameras are used for depth mapping in a variety of scenarios [2, 6, 8, 10, 13]. To measure the depth of a scene, a C-ToF camera sends a light signal towards the scene, then measures the phase shift of the reflected light. Unfortunately, this simple design means that when multiple cameras are present, they interfere; for example, one camera may receive the reflected light from a different camera instead of its own reflected light, meaning the first camera will record an incorrect depth measurement. This issue is called multi-camera interference (MCI) and is an important open problem in the space of C-ToF cameras.

2.2 Addressing Multi-Camera Interference

Previous work focused on eliminating the multi-camera interference using time-division multiplexing by ensuring that every camera operates at a different time period to avoid interference [18]. Inconveniently. this approach requires synchronization among cameras and a central device to control them. Another similar approach involves having unique modulation frequencies for each camera to avoid interference [14, 19]. However, due to the limited availability of usable frequencies, only a few TOF cameras can work simultaneously, making the network less scalable.

The pseudo-noise modulation techniques proposed in [3] [4] [19], focus on making the reflection signals from other cameras uncorrelated to avoid multi-camera interference. However, these methods require upgrading existing TOF cameras to utilize the sinusoidal modulation technique. As interference signals are related to the non-interference signals, it is difficult to recover the correct signal from the multi-camera interference.

This paper [12] uses Least Squares Estimation to estimate the non-interfering component. However, these approaches do not aim to remove the DC interfering component, which results in higher noise. [11] paper uses Stochastic Exposure Coding where total exposure time is divided into slots. In each slot, the camera and source are turned on with a random probability. While this paper eliminates both AC and DC component of interference, still uses

sequence of random probabilities. Our proposed method builds upon SEC approach by intelligently mitigating collisions using machine learning.

2.3 Related Work Involving Machine Learning

Machine learning (ML) has emerged as an universal solution to many open challenges that are difficult to solve using traditional engineering approach, especially for networking under complicated, wireless air space. Most similarly to our work, prior work has proposed incorporating ML in MAC protocols, either by identifying the current MAC protocol in use [16], using ML for spectrum management in wireless networks [17], or using ML to improve MAC protocols (e.g., using Q-learning with Slotted Aloha) [1]. The latter is the most similar to our project's goals, but our scenario is not quite comparable to that of a MAC protocol and is much more complicated. For instance, to train a model which slots are best to fire for Slotted Aloha, one can use the success or failure of transmissions in order to fire. However, in our scenario, we must be able to interpret RGB frames as input and must take into account the accuracy of the depth measurement, neither of which is trivial.

In the realm of ToF cameras, machine learning has mostly been applied for automatic calibration of consumer cameras [7] or improving the accuracy of depth measurements [9, 15]. To our knowledge, no prior work has attempted to use machine learning to mitigate MCI in C-ToF cameras. Our initial definition of the problem was quite simple: train a single model that, when replicated across multiple actors, minimizes the total loss across all defined scenarios across all actors, thus requiring the actors to collaborate. In our research, we did not encounter any archetypal problems that matched this approach. While "Collaborative Machine Learning" does exist, this is usually referring to Federated Learning, where each instance of a model is trained separately and the updates to the model are uploaded to some central server and shared across all instances. Critically, in our case, it is impossible for a single instance of the model to determine whether it has improved in isolation. "Collaborative" in the case of federated learning means that multiple instances are training at the same time to learn more quickly than if they did not share their training updates. It does not cater to the possibility that collaboration between the running instances of the model is necessary to achieve the optimal solution. While we believe that federated learning can be used to improve our approach, it is orthogonal to the fundamental challenges of our solution.

3 PROPOSED METHOD

3.1 Method Overview

The high-level pipeline of our solution is as follows:

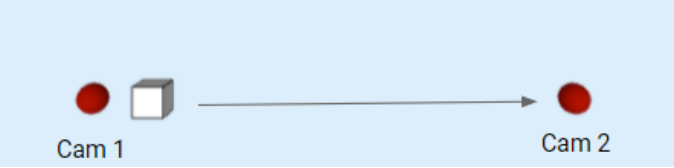
- RGB camera takes in a scene frame by frame
- Trained AI blackbox identifies whether the camera should fire on each frame
- This fire decision is sent to the ToF camera, which fires directly according to instruction from the blackbox

When running, the instances of the model are not able to communicate explicitly - at least, not synchronously. While C-ToF sensors are capable of firing thousands of times per second, for the sake of this project, we chose to assume that one video frame corresponds to one fire event. The frame rate is assumed to be 30 frames per second (fps). Thus, the length of a fire event is approximately 33ms. Once a viable solution to this simplified version is found, it could be expanded to have the blackbox output a probability of firing for all slots in the time between captured frames, or possibly a vector of fire instructions for each of the slots between frame captures.

3.2 RGB Data for Training and Testing

To facilitate data creation in an automated fashion while being able to access ground truth distance values, we decided to use a programming environment, namely Processing, and started with a simple 2-camera setup. As shown in an overview perspective in Fig. 1, the red balls represent camera locations, with the white box as a moving object in the horizontal direction. A list of ground truth distance values between cameras and object is generated after the scene exporting process. This scenario serves as a proof-of-concept application for our AI model training, however, it is limited in the number of interfering cameras, object's moving speed and camera locations.

Figure 1: Training setup Two Camera

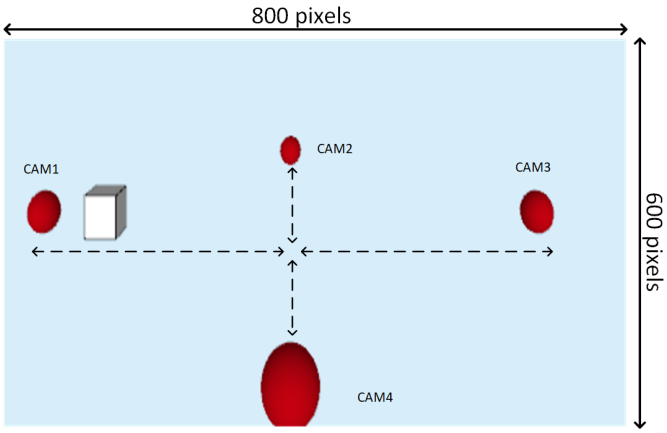


To further expand the scale of our datasets, we built upon the existing 2-camera scenario and introduced a more sophisticated model with 4 cameras and one single moving object in Fig. 2. The final setup has a dimension of 800 x 600 pixels in each direction. To diversify the scene contents, we implemented an algorithm to employ random seeds for camera locations so that they are placed differently in each scenario in both the horizontal and vertical directions. To avoid situations when one camera might block another camera's view under this randomized mechanism, specific boundaries are set for each camera's random seed to minimize confusions. For instance, as shown in Fig. 2, the possible vertical locations for Camera 1 is enforced to be between Camera 2 and Camera 4. Its horizontal location is also restricted so that the moving object is always in front of itself, while not crossing the line between Camera 2 and Camera 4. Based on camera locations in different scenarios, the moving object travels from left to right for roughly 100 frames and stops before hitting Camera 3. Additionally, we also vary the speed of the moving object to diversity the datasets. For the training purpose of this project, a total of 20 scenarios are created with recorded motion pictures for all 4 perspectives.

The definition of training data and test data is unusual in our situation. The goal of this project is not for the actors to be able to identify and avoid collisions in scenarios they have not seen before, but to be able to identify previously seen scenarios, and use the previously computed optimal distribution of channel time. We would expect, for the time being, a human to create and label

certain scenarios, such as a highway merge or a 4 way stop sign, and then create several permutations of this using a randomizer as described above. The ability for the model to infer the correct firing instruction for a frame thus hinges on its ability to identify both which labeled scenario it is participating in, and which actor it is. In the future, we would expect to be able to collect this data from real cars and categorize these scenarios using unsupervised techniques.

Figure 2: Training setup Four Camera



3.3 AI Model

After setting scenario parameters in Processing, the perspective of each individual camera is captured with a dimension of 800 x 600 pixels, with motion pictures generated based on these frames. To further improve the overall training efficiency and convenience with the amount of data available, the data frames from all 4 cameras are flattened in to a one-dimensional array. Then, the frames in this array are re-organized in an alternating order, meaning that each training slice contains one frame from each camera's perspective. In this way, the loss function is able to account for all actors in the scenario when determining the loss. We then implemented a computer vision algorithm to extract features from the RGB values based on motion picture frames, and map them to binary firing orders (0 as silence, 1 as firing) with ground truth values from earlier steps in data creation. Results in this step are then channeled into a loss function algorithm to iteratively minimize errors between estimated distance and ground truth distance, which constantly updates firing orders. Once the accuracy of our model under training converges to an acceptable level, the iterative process stops and a final firing order is achieved.

3.3.1 Approach. The most controversial part of our approach is that it requires centralized, or at least, federated learning. It cannot be done in a decentralized fashion. It also requires supervision, both for the labels of scenarios, and the ground truth depth that the C-ToF camera should read during each frame. The question is whether a decentralized approach would be able to converge upon the optimal solution. The fundamental issue is that for an instance that has no knowledge of other instances, it is always beneficial to fire. C-ToF sensors can detect collisions, and this can be used as feedback for a reinforcement learning algorithm, which is can train in a decentralized fashion. Depth estimation accuracy is what the model is trying to optimize. For each frame, the blackbox can choose to decide to fire or not fire. If it chooses to fire, and the probability of a collision is inferred by the blackbox to be p_c, then the probabilty of a successful read is $1-p_c$, which is usually non-zero. Sadly, the probability of an successful read if the blackbox chooses not to fire is always 0. Thus, the blackbox will never choose to remain silent. We go into further detail about how a reinforcement learning algorithm could be designed to overcome this issue in the Future Work section. Admittedly, our team was unfamiliar with AI techniques, so perhaps our judgement is from a place of ignorance. We will not make the claim that our AI approach is optimal. This solution is simply what came to us first based on our rudimentary understanding of AI. Perhaps our approach is better seen as an investigation into the potential of AI to communicate implicitly using observation of the environment.

To further illustrate our point, consider the example where two cars come to a 4-way stop simultaneously. In this scenario, human drivers implicitly understand which of them is counter-clockwise relative to the other, and thus has priority. There is no central authority that determines who is where. The central authority was only responsible for explaining rules to the human drivers during training (driver's education). There is also no communication between drivers. This model of centralized training and decentralized execution allows for conflict-free use of the road. While coming up with rules that allow for efficient use of the channel is important, we must first ensure that actors have the ability to process their environment and know when these rules should be applied.

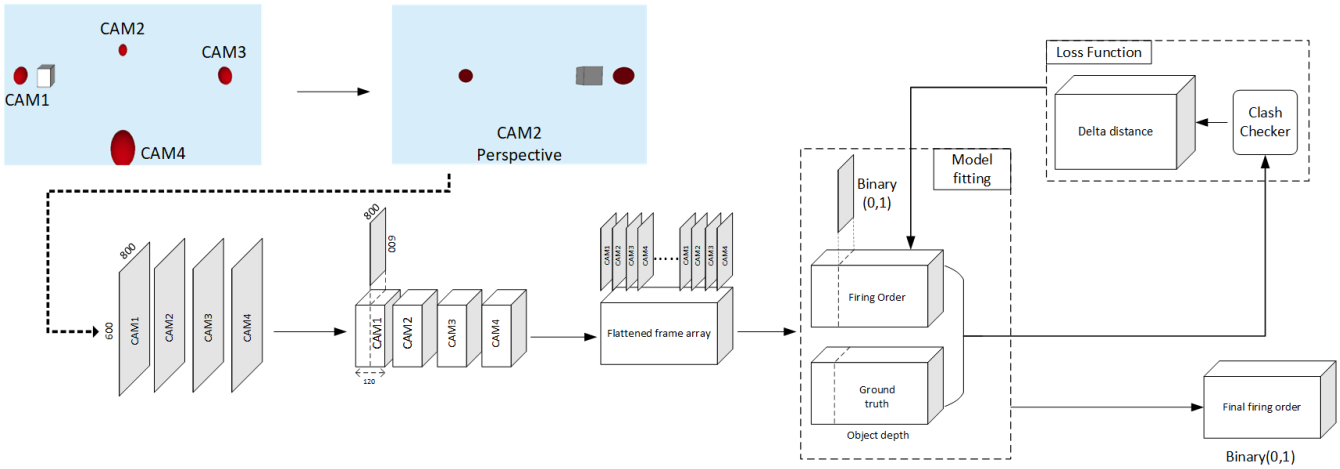
We used a sequence of frames as an input to the model because this allows the model to extract features that can only be seen in a sequence. For example, the direction and speed of motion or the appearance or disappearance of some object. We also experimented with using depth values as an input to the blackbox. The C-ToF sensor outputs depth values independently of the blackbox, and assuming minimal interference, previous depth values can be relied upon by the blackbox as an additional indicator about the scene.

3.4 Implementation

3.4.1 Modifying the SEC Simulator. The SEC MATLAB Simulator calculates the estimated depth values with a fixed ground-truth value as an input. The number of slots with collisions is calculated using a series of binary random probability values for each camera, and based on un-collided slots, the interference amount is calculated. The following modifications are done in the simulator code to work according to our requirements.

- The simulator code has been modified to consider a matrix of different ground truth values per camera per frame to calculate corresponding depth values. A graph showing difference between the ground truth value and the calculated depth value is plotted.
- As part of our interface, ground truth values and probability values are fed into the simulator as input with estimated ground truth values as output in the form of a .csv file used for training the model.

Figure 3: System overview



- The proposed simulator interface served as a proof-of-concept module in our initial prototype's workflow. It was relegated to be a tool for evaluation because the simulator ran slowly, did not penalize collisions enough to effectively train the AI, and the random noise it simulated was not conducive to efficient learning.

3.4.2 AI Framework. We used Tensorflow and Keras in Python to implement our solution. Our dataset consisted of slices of consecutive RGB frames. The input shape of our model would thus be (dataset length, slice size, height, width, 3). We used a series of Conv2D, BatchNormalization and MaxPooling2D layers for the vision section of our AI stack. LSTM (Long Short Term Memory) is also used as a type of Recurrent Neural Network (RNN) to process the frames in the slice sequentially. Currently, our prototype uses a stateless RNN, which means that with every datapoint of s frames, the neural network must process all s frames. Switching to a stateful RNN would allow the model to roll up result of processing the previous s-1 frames, and thus avoid costly recomputation. We also used Dense layers, as is typical for deep learning models, and Dropout layers to prevent overfitting. We attempted to use Adam gradient descent to optimize our model.

3.4.3 Loss Function. As mentioned above, we originally intended to use the results of the simulator in our loss function. To replace it, we implemented an extremely simple version of the clash checker in the simulator as follows:

```
for each (scenario, actor, frame):
    if fire = 1 and collision = false:
        depth_estimate := ground_truth
    else:
        depth estimate := 0
```

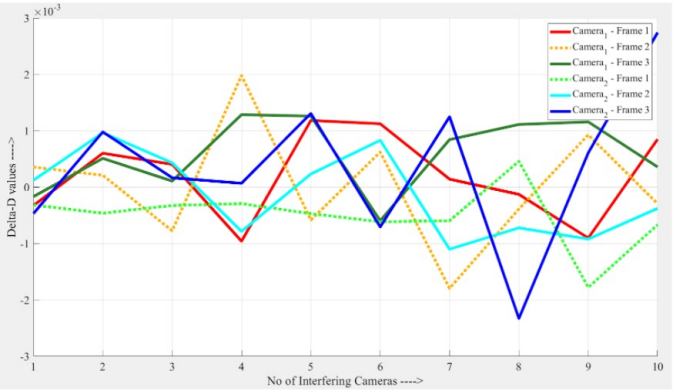
The loss function would simply be the mean squared error between the computed depth estimate and the ground truth. This works decently well as a *metric* for determining whether we've achieved our goal of avoiding collisions. The average delta between estimated distances and ground truths will obviously be lower in the case of fewer collisions, and greater in the case of more collisions. Unfortunately, we found out too late that the loss function must be differentiable for gradient descent to work. This is obvious to those familiar with machine learning, of course.

We then decided to write a function that approximated this clash checker's output but was still differentiable. We attempted to simply multiply the firing instruction by the ground truth to determine the estimated distance. For example, if the actor who should fire outputs 1, their estimate would equal the ground truth. If the actors who should not fire output 0, their estimate would equal 0. On its own, this results in all actors firing at all times, of course. We then decided to penalize the model if more than one actor fired in a single frame. This resulted in the actors with the smallest deltas to fire with the smallest probability - a major result. This meant that the model was indeed capable of having one instance of the model sacrifice itself for the greater good in a given situation. Unfortunately, the models that had small deltas only were so because the ground truth distance itself was small. 0 minus a small distance creates a smaller penalty than 0 minus a large distance. Once we normalized the penalties according to distance, the models settled on the equilibrium state, which is to fire randomly with probability 1 /number_of_cameras. A great result as well, since this is the rational equilibrium solution in a state of non-cooperation, and approximates the behavior of SEC. It's obvious that we came somewhat close with the implementation, but the loss function must be tuned further to give priority to those who expect to see the biggest information gain from firing.

One complication that held our implementation back was that we could not figure out how to use state when training our model. We came into this assuming that if an actor is unable to fire, it would simply assume that the estimated distance of the current frame is the same as the estimated distance of the previous frame. This could be the ground truth distance from 1 frame ago or 100 frames ago - it depends on when the last fire event for this particular actor was. Without training each frame in each scenario in chronological order and saving the last estimated distance for each actor as state, it would be impossible to determine the last estimated distance. We believe that if this problem were solved, our loss function would

work, because more static actors would have less to lose and thus incur less of a penalty than more dynamic actors whose ground truth distance is changing very quickly. The model would optimize to give priority to the actors who are more dynamic and we would achieve our goal.

Figure 4: DeltaD vs No of Interfering Cameras



4 EVALUATION

4.1 SEC

We evaluated the modified simulator interface in two categories: delta distance and standard deviation caused by interfering cameras. As shown in Fig. 4, we looked at the difference between estimated depth and ground truth depth based on a 2-camera scenario for the first 3 frames. The results demonstrate that the overall error is less than 2 mm for all frames with up to 10 interfering cameras. In Fig. 5, we also observed that the error standard deviation is less than 2% in for the first 4 frames in our final prototype of 4 cameras. Based on these two evaluation results, combined with other concerns mentioned in Section 3.4, we decided to remove simulator response from our model due to its ineffectiveness for penalizing the model, which hinders training progress.

4.2 Fingerprinting Scenarios

As shown in Fig. 6, we were somewhat successful in getting each instance of the model to recognize in just a few frames which scenario it was observing, and from which perspective. This lends credence to the idea that each actor in a scenario can implicitly understand what role it should play in its environment to ensure optimal use of the channel. It's not necessarily important that the model be able name which scenario and which perspective each frame-slice belongs to. These human-prescribed names are extremely fickle. One could imagine the existence of two permutations of a scenario that are nearly identical, but have the names of the cameras in a different order due to the randomizer. This would make it impossible to name the camera correctly. Thus, it would take some sort of unsupervised algorithm to cluster similar scenarios and perspectives together.

4.3 CV-based depth estimation

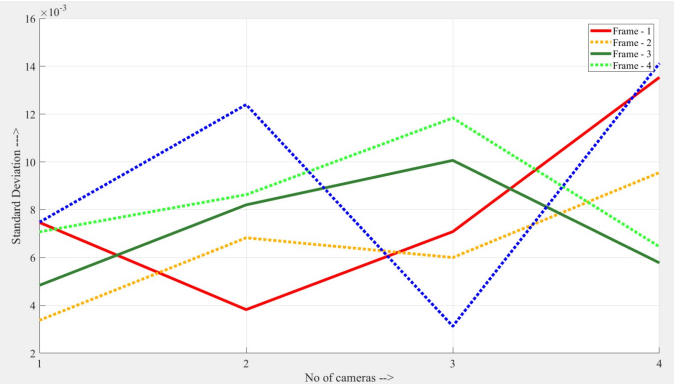
We investigated how effective computer-vision based depth estimation would be. It can be said that the ground-truth depth of each frame is a type of fingerprint, and our model's ability to map a slice of RGB frames to its associated depth would indicate that the computer vision stack is able to extract features out of the frames effectively. We were able to achieve a training accuracy of 90.5% using 4-frame slices. While we understand that training error is not as meaningful as test error, this bolsters our impression that the model is able recognize scenes via vision.

4.4 Using RGB frames and depth as input

Since the C-ToF sensor is capable of accurately estimating depth, we trained a model using both 8-slices of RGB frames and 8-slices of depths as input to predict the scenario name and actor name. This was our most complex prototype. It achieved a scenario classification training accuracy of 18%, which is almost 4x better than guessing randomly. It was also able to guess actor names with 66% accuracy. However, this was only trained for 10 epochs of approximately 8 minutes each. The computer used had a 6-core Intel i7-10750H, 16GB of RAM, and an Nvidia 2070 Max-Q GPU with 8GB of VRAM. Despite the power of the computer, we had to train the model in batches of 2 datapoints. Making the batches any larger exceeded the memory of the GPU. It is unclear whether better accuracy could be achieved with more training time.

The inference time, in batches of 1 datapoint (consisting of 8 RGB frames and 8 depths) was 50ms. This would support a frame rate of 20fps. We believe that there are significant performance gains to be made from switching to a stateful RNN as mentioned above, as this would reduce the computation demand by approximately 7/8.

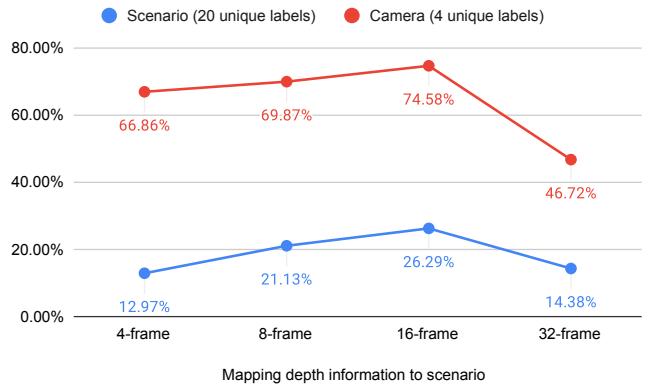
Figure 5: Original SEC Simulator Result



5 FUTURE WORK AND CONCLUSION

There are 20 scenarios in total used for training in the final prototype. Though various speed and camera setup are considered, the datasets is still limited to the 4-camera scenario with relatively large space between cameras and linear trajectory of one single, moving object. Future steps in this area should include a large scale network with many more cameras and moving objects travel in

Figure 6: This shows the relationship between size of slice and accuracy of determining which scene the model is looking at. Note that this is only when looking at past depth information, not the RGB frames. This is for a fixed amount of training time (50 epochs). This justifies our decision to use an RNN that processes a sequence of frames. It also proves that scenarios and perspectives within each scenario each have somewhat unique fingerprints.



stochastic directions. The motion picture creation should also be enhanced to deliver a fully end-to-end, automated workflow to improve fabrication efficiency.

Another approach would be to use reinforcement learning, where each instance of the model trains in isolation and uses feedback to improve itself. If we assume that the C-ToF sensor is capable of detecting collisions after they happen, this could be used by the blackbox to map each scenario to its probability of collision, which would factor into the decision to fire. A reward function could be designed to compensate an actor for not firing. This reward would be proportional to the gains of the actor who ought to fire in the scenario. An actor would weigh its two options: use the last estimated depth and incur whatever penalty arises from the delta, but be compensated by the reward function, or attempt to fire, which would either result in a successful read or a collision and a penalty both from the delta and from the reward function. This solution does seem more attractive than our supervised approach, in hindsight. In our defense, we would argue that having a reward function that takes global gains into account and incentivizes collaboration still requires some sort of centralization. In other words, we are unaware of a solution to this problem that totally bypasses the need for collaboration and centralization.

Another alternative that would solve our issue with the non-differentiable loss function would be to use a genetic algorithm. Our cursory research shows that genetic algorithms do not suffer from this issue.

Surely, we've made it clear that that this project has raised more questions than answers. We set out to solve what initially seemed like a simple problem, but quickly found that it was beyond our expertise. We hope that our reasoning and investigation can serve to further discover the potential of implicit communication using AI. While it may not be the most effective solution for C-ToF collision avoidance, this serves as an interesting springboard towards what we believe could be an unexplored realm.

6 CONTRIBUTIONS

Yoganand is on the sub-team with Yuchen, and he is mainly responsible for the modifying the simulator, creation of datasets and working on some aspects of related work on multi-camera interference. His contributions include: As an initial step, worked with Yuchen and the original authors to understand the underlying working of the current simulator and tried to modify different simulator settings. As a part of simulator modification, worked with other team members to come up with and develop possible solutions for the system interface to use the upgraded simulator with the AI Blackbox for training and depth calculation. Using the initial model developed by Sophie, worked with Yuchen to implement a randomized algorithm by changing the parameters different permutations in a multi-camera scenario and created data set of total 20 such scenarios to be used for training the model. Participated in design discussions with the machine learning team to finalize the interface.

Yuchen is on the sub-team with Yoganand, and he is mainly responsible for the simulator modification, dataset creation and system level, architecture design of the project. His contributions include working with Yoganand and the original authors to understand the fundamental mechanism of the existing simulator. He attempted to tune different parameters of the simulator to prove initial prototype assumptions. He worked with the rest of the team to propose and implement tangible solutions to create necessary infrastructure for the modified simulator to be used with the machine learning model. He adapted an initial prototype from Sophie and worked with Yoganand to expand it to 20 learning-ready scenarios. He explored and implemented an automated motion picture creation workflow based on generated frames. He worked with Yoganand to implement a random algorithm to set up multi-camera scenarios. Lastly, he participated in the system design discussion with the machine learning team, and created system-level workflow illustrations.

Saurabh worked on the software team. His contributions include coming up with the project idea, managing the the project, implementing some of the AI stack, and collecting the evaluation results.

Sophie worked with Saurabh on the software team. Besides producing the original RGB scene in Processing, she designed a rudimentary reinforcement learning algorithm as a proof-of-concept early in the process. This prototype used computer vision to analyze an RGB movie, gather information about the speed, direction, and size of the moving object, and predict whether to fire based on a series of trained parameters. She also designed a backup AI model which took in individual frames along with an ideal firing sequence (e.g., alternating 0s and 1s for a two-camera scenario) and attempted to reproduce the correct sequence. Finally, she contributed to whole-group discussions about the architecture and implementation of our solution.

REFERENCES

[1] Noor Zuriatunadhirah binti Zubir, Aizat Faiz Ramli, and Hafiz Basarudin. 2017. Optimization of wireless sensor networks MAC protocols using machine learning; a survey. In *2017 International Conference on Engineering Technology and Technopreneurship (ICE2T)*. IEEE, 1–5.

[2] Pia Breuer, Christian Eckes, and Stefan Müller. 2007. Hand gesture recognition with a novel IR time-of-flight range camera–a pilot study. In *International Conference on Computer Vision/Computer Graphics Collaboration Techniques and Applications*. Springer, 247–260.

[3] Bernhard Buttgen, Felix Lustenberger, Peter Seitz, et al. 2007. Pseudonoise optical modulation for real-time 3-D imaging with minimum interference. *IEEE Transactions on Circuits and Systems I: Regular Papers* 54, 10 (2007), 2109–2119.

[4] Bernhard Buttgen and Peter Seitz. 2008. Robust optical time-of-flight range imaging based on smart pixel structures. *IEEE Transactions on Circuits and Systems I: Regular Papers* 55, 6 (2008), 1512–1525.

[5] Raghu Changalvala and Hafiz Malik. 2019. LiDAR data integrity verification for autonomous vehicle. *IEEE Access* 7 (2019), 138018–138031.

[6] Yan Cui, Sebastian Schuon, Derek Chan, Sebastian Thrun, and Christian Theobalt. 2010. 3D shape scanning with a time-of-flight camera. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. IEEE, 1173–1180.

[7] David Ferstl, Christian Reinbacher, Gernot Riegler, Matthias Rüther, and Horst Bischof. 2015. Learning Depth Calibration of Time-of-Flight Cameras.. In *BMVC*. 102–1.

[8] Jan Fischer, Benjamin Huhle, and Andreas Schilling. 2007. Using Time-of-Flight Range Data for Occlusion Handling in Augmented Reality. *IPT/EGVE* 109116 (2007).

[9] Ying He, Bin Liang, Yu Zou, Jin He, and Jun Yang. 2017. Depth errors analysis and correction for time-of-flight (ToF) cameras. *Sensors* 17, 1 (2017), 92.

[10] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. 2014. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. In *Experimental robotics*. Springer, 477–491.

[11] Jongho Lee and Mohit Gupta. 2019. Stochastic exposure coding for handling multi-tof-camera interference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 7880–7888.

[12] Lianhua Li, Sen Xiang, You Yang, and Li Yu. 2015. Multi-camera interference cancellation of time-of-flight (TOF) cameras. In *2015 IEEE International Conference on Image Processing (ICIP)*. IEEE, 556–560.

[13] Stefan May, Bjorn Werner, Hartmut Surmann, and Kai Pervolz. 2006. 3D time-of-flight cameras for mobile robotics. In *2006 IEEE/RSJ International Conference on Intelligent Robots and Systems*. Ieee, 790–795.

[14] Dong-Ki Min, Ilia Ovsiannikov, Yohwan Noh, Wanghyun Kim, Sunhwa Jung, Joonho Lee, Deokha Shin, Hyekyung Jung, Lawrence Kim, Grzegorz Waligorski, et al. 2013. Pseudo-random modulation for multiple 3D time-of-flight camera operation. In *Three-Dimensional Image Processing (3DIP) and Applications 2013*, Vol. 8650. International Society for Optics and Photonics, 865008.

[15] Malcolm Reynolds, Jozef Doboš, Leto Peel, Tim Weyrich, and Gabriel J Brostow. 2011. Capturing time-of-flight data with confidence. In *CVPR 2011*. IEEE, 945–952.

[16] Margaret M Rooney and Mark K Hinders. 2021. Machine Learning for Medium Access Control Protocol Recognition in Communications Networks. *IEEE Access* 9 (2021), 110762–110771.

[17] Yaohua Sun, Mugen Peng, Yangcheng Zhou, Yuzhe Huang, and Shiwen Mao. 2019. Application of machine learning in wireless networks: Key techniques and open issues. *IEEE Communications Surveys & Tutorials* 21, 4 (2019), 3072–3108.

[18] Felix Wermke, Thorben Wübbenhorst, and Beate Meffert. 2020. Optical Synchronization of Multiple Time-of-Flight Cameras Implementing TDMA. In *2020 IEEE Sensors*. IEEE, 1–4.

[19] Refael Z Whyte, Andrew D Payne, Adrian A Dorrington, and Michael J Cree. 2010. Multiple range imaging camera operation with minimal performance impact. In *Image Processing: Machine Vision Applications III*, Vol. 7538. International Society for Optics and Photonics, 75380I.