Collecting an event-based dataset for a quadrupedal robot.

D. van Loon (student number 5346894), E. van Huffelen (student number 5411114),
J. van Brakel (student number 5323215) & R. Makinwa (student number 5270677)
Delft University of Technology, Faculty of 3mE, Mekelweg 2, 2628 CD Delft
Supervisors: MSc. Chuhan Zhang, Dr. Jiatao Ding & Dr. Cosimo Della Santina

Abstract

In the last few years, event-based cameras have been explored for use in computer vision, as these cameras show significant advantages over normal frame-based cameras, including: low latency, high dynamic range, robustness to noise and high temporal resolution. A lot of data has come with numerous research papers, but unfortunately, to the best of our knowledge, a key aspect is missing; no such datasets are taken from the point of view of a quadrupedal robot.

In this paper, the methodology is discussed of collecting an event-based dataset using the point of view of a quadrupedal robot, hoping this will promote more research to be done using event-based cameras on robots. This dataset is collected in ten different scenes using a Unitree Go1 robot with an Inivation DAVIS 346 event-based camera, where a large amount of sensory data from both the camera and robot is available in the dataset. This allows for the training of a complete navigational neural network for quadrupedal robots. The dataset is provided in binary format (rosbag) with the addition of five camera streams from the Unitree Go1 robot in WebM or MP4 format. This paper provides a further overview of all available data, the collection and processing method and any recommendations for the usage of this dataset.

1 Introduction

In previous years, standard, frame-based cameras have been dominating the field of computer vision. Frame-based cameras have been widely used in autonomous vehicles, drones, sports cameras, and robots alike. The performance of the robots partly depends on the frequency with which the perception system takes in data as well as how quickly that data is processed. Poor latency of a perception system can set a hard boundary on the performance that the system as a whole can achieve. Therefore we believe that using event-based or neuromorphic cameras can be a valuable alternative for robots. Event-based cameras only capture changes in brightness, or events, instead of frames at a set interval like normal cameras. Because of this event based cameras have very low latency, high dynamic range and temporal resolution. This means that event-based cameras can capture even in very dark situations, where greyscale and RGB cameras would fail, and capture changes at a much higher speed than greyscale and RGB.

A potential application where event-based cameras would be beneficial is for the navigation of quadrupedal robots. Because they operate in rapidly changing conditions, the aforementioned benefits of event-based cameras would allow quadrupedal robots to navigate better in various situations. A full and comprehensive dataset allows neural networks to be trained upon it. The trained neural networks can be used for the navigation of quadrupedal robots, because they allow us to recognise the patterns in the relatively new type of data stream generated by the event-based cameras. Other event-based datasets are being created and open-sourced [1]. However, these datasets often don't account for various weather conditions and are generally taken from a handheld position. Moreover, so far no event-based camera dataset has been found to be captured from a quadrupedal robot. By publishing this dataset, further development in quadrupedal robot navigation will be possible, finally making this new technology available for use in common robots. Our goal is to collect a full dataset from the event-based camera and the quadrupedal robot using Robot Operating System (ROS) and describe the methodology to use and recreate it.

There are several datasets that have covered event-based datasets, one of these is "*The Event-Camera Dataset and Simulator: Event-based Data for Pose Estimation, Visual Odometry, and SLAM*"[1]. The dataset created in that paper contains all the data from the event-based camera; the events, images, IMU and the groundtruth, on 13 different scenarios. Our dataset contains the same information from the event-based camera, but also contains all the sensors from the Unitree Go1.

The purpose of this work is to describe a methodology to collect a full event-based dataset, which combines the data from the event-based camera and the data for the quadrupedal robot using ROS. In section 2 the general overview of both the camera and robot used will be provided. section 3 will describe how the data from all the sensors is captured, and what pre-processing steps were applied. The actual dataset will be covered in section 4. This study has been conducted as a part of the Bachelor End Project for Mechanical Engineering students at the TU Delft.

2 Background

For this project, the Dynamic and Active-pixel Vision Sensor (DAVIS) 346 (see Fig. 1), fabricated by IniVation, was used [2]. Event cameras capture events instead of full pictures of every pixel at set time intervals. An event contains 4 pieces of information, the x and y coördinates of the pixel, the time t the event occurred and the polarity p. This results in the following data structure: $e_i = [x_i, y_i, t_i, p_i]$. Polarity indicates whether the brightness change was positive or negative. The response of these cameras goes down to the microsecond level, which makes it effective for any highly responsive system, such as a robot. While the resolution of the DAVIS 346 is not impressive with its 346x260 pixels, it does capture a grayscale image with its Dynamic Vision Sensor (DVS) and Active Pixel Sensor, all while keeping energy consumption to a minimum. The advantage of this type of camera is the incredibly low latency, high temporal resolution, and no redundant information (so small data sizes). Because of the high temporal resolution, the event camera can operate indoors and outdoors without the need to change parameters, such as exposure. This is especially useful when entering or exiting buildings, as traditional cameras require a second for the exposure to adjust, making the robot operator or AI blind.

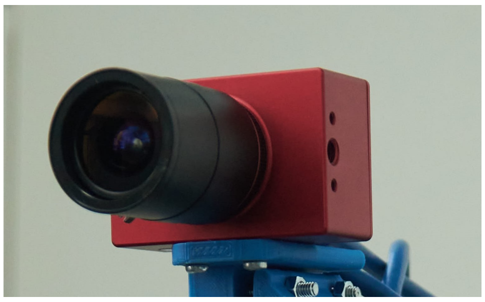


Figure 1: The DAVIS 346 camera mounted to the Unitree Go1 robot.

The Unitree Go1 EDU quadrupedal robot, as shown in Fig. 3), has been employed in a different research context, where it was modified by adding a backpack on top of the robot. The quadrupedal has numerous sensors on board. The sensor suite includes a combination of vision, depth perception, and motion sensors, allowing for precise mapping and localization capabilities. Its motors can also be used for force measurements, enabling the robot to walk on different terrains while keeping balance. Furthermore, the Go1 integrates robust communication capabilities, including Wi-Fi and Ethernet, enabling seamless connectivity and data exchange with other devices or systems. This extensive array of sensors empowers the Unitree Go1 to perform complex tasks, including autonomous navigation, dynamic obstacle avoidance, and interactive engagement with its surroundings.

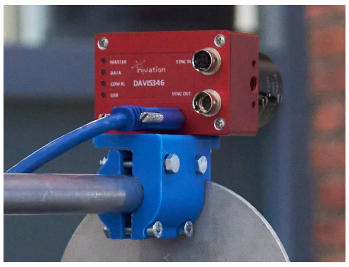


Figure 2: The camera mount clamped to the robot backpack with the event-based camera on top.

To combine the Go1 robot and the event-based camera, a mount needed to be fabricated to act as a link between the backpack and the mounting hardware of the camera. Our team did this by utilizing an FDM-printer and some simple CAD. The backpack of the robot had a pickup handle, made from a solid aluminium rod. In combination with a clamp around the edge plate, the mount could be clamped around the rod and plate, providing a secure connection to the robot and camera. The mount is shown in Fig. 2.

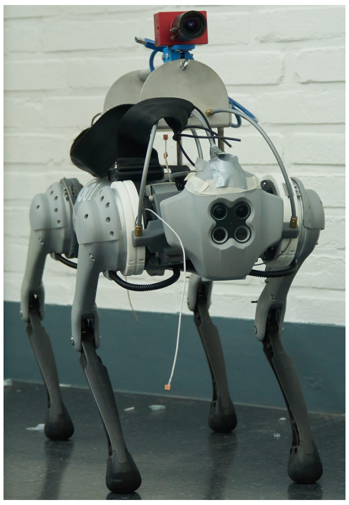


Figure 3: The Unitree Go1 robot, with the DAVIS346 mounted on top.

To successfully produce an event-based dataset useful for training a neural network for navigation, some requirements need to be set. Capturing events from the event-based camera with minimal to no packet loss is a critical aspect to consider in this context. Furthermore, most if not all sensor data should be included in the dataset. In Table 1, all the sensors are shown. Furthermore, the sensors that are minimum requirements for the dataset to be useable are noted. In addition, reasons are provided for why they are imperative for the dataset to be useable. The gathered dataset will be evaluated on the following criteria: Amount of sensors captured, total length and number of different scenes.

3 Methodology

In this section, our method is described for collecting an event-based dataset for quadrupedal navigation using a modified Unitree Go1 robot and an IniVation DAVIS 346 event-based camera. As will be discussed in the next chapter, the resulting dataset consists of data from the robot and from the camera. From both the robot and the camera data was collected from all sensors present, which can be found in Table 1. The following subsections describe our method for collecting the different sensors.

For the dataset, a few different scenarios have to be used. By choosing the scenes in a clever way, all the different advantages of event-based cameras can be shown off. The dataset should, furthermore, provide scenarios where either a similar camera and/or robot could be used in the 'real' world. This dataset could be used to train other robots to use the event-based camera for navigation. The dataset needs to cover multiple different options for terrain, human presence, lighting conditions, amount of edges and distance to objects. The captured scenes are covered in section 4.

Data source	Sensor	Minimum	Reasoning
DAVIS346:	Events	X	Bare minimum for dataset to be useable
	Greyscale	X	To still have vision even with exclusively static objects
	IMU	X	Alternative if IMU from Go1 is not included, but less ideal
	Camera info		Contains information on lens distortion and exposure, not imperative for dataset to be useable
Unitree Go1:	5 RGB cameras		Good addition, but not minimum requirement as DAVIS346 provides video
	IMU	X	Needed for groundtruth
	Velocity	X	Together with IMU data groundtruth can be found
	Position		With Go1's IMU and Velocity, groundtruth can be found without position
	Point cloud	X	Depth information is needed for navigation
	Range		Point cloud gives similar data, but more is extensive
	Foot force		Not necessary for groundtruth
	Joint angles		Not necessary for groundtruth

Table 1: All sensors on Go1 and DAVIS346, and which are minimum requirements

3.1 Collection processes

In order to capture the data, ROS was used. The robot was running the ROS master node on ROS Melodic and an external machine connected via Wi-Fi to it running ROS Noetic. This mixing of machines and ROS versions was a deliberate decision to overcome some hardware and software challenges.

Hardware challenges As ensuring a complete event-based dataset was the first priority, some requirements on the hardware setup were made.

Firstly, it is recommended that a rosbag of a camera stream is captured on the same machine the camera (node) is connected to, due to network bandwidth requirements. Secondly, a USB 3.0 port is required to prevent a bottleneck in the event-based camera data stream. Lastly, the recording machine needs enough RAM and storage speed to handle the data rate of the recording.

The onboard computer of the Unitree Go1 did not meet these requirements, causing the need for another machine. A Raspberry Pi 4B or similar was considered, since they could be mounted to the robot. However, these single-board computers are still difficult to obtain, since the chip shortage of 2020 [3]. In the end, carrying a laptop, near the robot, proved to be the easiest solution, while still ensuring no bottleneck was present.

Software challenges Some software challenges were also present in the system. Foremost, due to unrelated ongoing research, the robot used an older firmware version and could not be upgraded. Due to this, version 3.4.2 of the `unitree_legged_sdk` [4] had to be used. This, in turn, meant that the ROS version of the robot is ROS Melodic, which is no longer supported. This comes with the added challenge of requiring the use of Python 2, which passed its end-of-life date of January 1st 2020 and differs vastly from Python 3.

To mitigate these software challenges, it was chosen to use ROS Noetic on the external machine. This was possible as they share the same communication protocol. Another benefit of this is ROS Noetic is supported and uses Python 3.8. Another advantage of recording on the external machine was that it allowed us to set an unlimited size of the message buffer, which prevents the exclusion of messages.

3.1.1 Event-based camera

In order to record the data of the DAVIS 346 into a rosbag, a ROS driver for such a device is required. For this, two methods were devised:

RPG DVS ROS The `rpg_dvs_ros` package is a ROS package, made by the Robotics and Perception Group of the University of Zurich and ETH Zurich [5]. This package provides the message types, viewers, and ROS drivers for various Event-based cameras. Under the hood, this package uses `libcaer` from Inivation to interface with the camera. However, using this package could increase the risk of packet loss based on the previous experiments.

Custom node using PyAER After some research, `pyaer` was found, which is a Python package which provides an interface for event-based cameras [6]. This Python package uses the same library (`libcaer`) to interface with the Event-based camera. It was considered to write a ROS driver using this library since the loss problem could have been related to the `rpg_dvs_ros/davis_ros_driver` node. However, it is notable that maintainers of PyAER recommend using the `rpg_dvs_ros` package when using ROS.

In the end, it was decided to use the `rpg_dvs_ros` package, and check if any package loss was experienced and track down its source. An attempt to write our own ROS driver using PyAER would have been made, in the case that the cause was a result of the implementation of the rpg_dvs_ros package. The full ROS network is shown in Fig. 4.

Since one frame full of events alone is about 1.1 MB, which can be sent an unknown number of times per second, the data flow from the events alone is already quite large. The option was explored to use a single-board computer (a Raspberry Pi 3B) to keep the robot mobile. Unfortunately, the single board computer struggled performance-wise, as the limited memory failed to compile any driver packages and the micro SD storage failed to keep up with the high data stream of the event-based camera. This required an external machine to interface with the event-based camera and record the rosbags.

The event-based camera driver was also calibrated, for better image quality of the greyscale image. Also, a lens distortion calibration was done with camera_calibration node. Auto-exposure was also turned on to allow for recording outside in greatly varying lighting conditions.

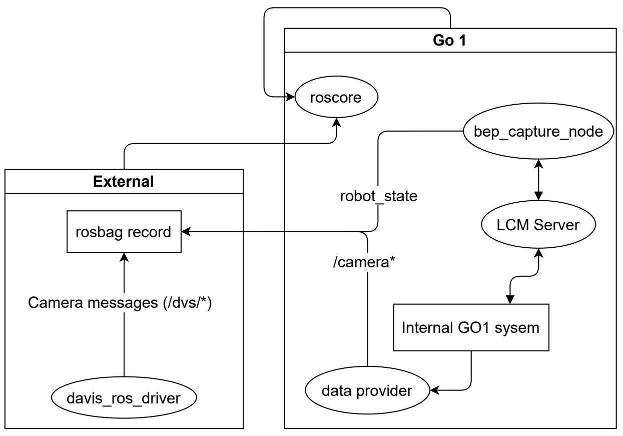


Figure 4: The ROS network

3.1.2 Collecting the robot state

In order to capture the internal state information of the Unitree Go1, a ROS node is needed to capture its internal messaging. Unitree has provided the unitree_ros_to_real ROS package to work with the robot [7], since the robot requires SDK version 3.4.2, as mentioned before, package version 3.4.0 is required.

This ROS package contains the required message definitions and conversion code, and some examples of controlling the robot with a predetermined plan from a ROS node. In this version of the SDK, the robot's communication goes via an LCM server[1] [8]. The initial attempts of creating a ROS node, which could publish the unitree_legged_msgs/HighState messages based on internal communication, were not successful. The full ROS network is shown in Fig. 4.

Therefore, a backup plan was devised in which the exposed Serial Debug port is read with a ROS node. This debug port sends similar state information, as can be found in a unitree_legged_msgs/HighState message. Implementation of this node was successful, which can be found in the unitree_serial package in our GitHub repository [9]. However, this solution is quite limiting since the debug information is only printed and therefore published at roughly 1 Hz, which is practically unusable compared to the velocity of the robot and the frequency of other subsystems.

After further attempts, a ROS node was created that could receive the data from the Go1 robot. This needed the following things to work:

- The LCM communication does not work over the bridged network, as a result, a ROS node which interacts with the LCM server must be run on the robot.
- The LCM server can only respond with a system state, so a message has to be sent before the state can be received.

The lack of documentation for the unitree_legged_sdk made creating the ROS node process difficult. As a consequence, it was unclear if the remote provided by Unitree would still work when communicating with the robot via an LCM server in high-level mode. To mitigate this, an attempt was made to make a controller using the LCM interface combined with the ROS teleop system. This was unsuccessful, nevertheless, it was discovered that the controller provided by Unitree kept working in this mode. So, a new capture node was created, which sends zero velocity commands to get a response from the LCM server, and publishes the state information as a unitree_legged_msgs/HighState message.

It is notable that the LCM server sometimes acted unstable, which resulted in a random crash. The origin of this was unfortunately untraceable, as the capture node will keep sending the last received robot state to the ROS network. As providing false data after the server crashes is undesirable, it was decided to trim the datasets accordingly. This trimming process is discussed further in section 3.2.3.

3.1.3 Collecting the robot's RGB cameras

As mentioned previously, the robot's firmware was not working as expected. This caused all camera data to be unavailable in ROS itself, which made it impossible to collect the RGB cameras the traditional way in ROS. To get around these issues, the RGB cameras were collected with a workaround. This workaround works independently of ROS unfortunately, which means that without the proper firmware, it was impossible to collect all camera streams into the rosbag. The workaround was designed to scrape the stream from a webinterface of the robot. This resulted in a Python script using Selenium and JavaScript injection to extract the WebM stream. The full script can be found in this paper's GitHub repository [9].

Using this script, five browser sessions are automatically booted up into the webinterface of the robot. When a duration is entered in the terminal, a custom script is injected into the browser to start saving the webstream. After the duration has passed, all five camera files are downloaded in a 464x400 resolution running at 20 Hz. An overview of the cameras plus the event-based greyscale camera can be found in Fig. 5. As the webstream used VP-9 encoding in a WebM file format, all camera recordings are provided in

[1]*Lightweight Communications and Marshalling* or LCM is a set of libraries for inter-program and/or inter-system communication.

w

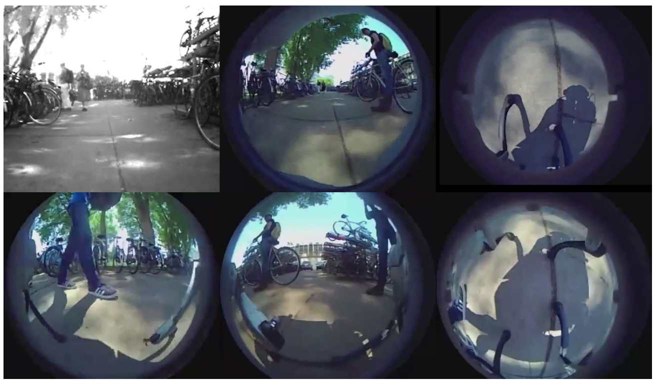


Figure 5: An overview of all 6 cameras mounted on the Go1.

WebM format. All clips are also provided in MP4 format with H.264 encoding for convenience, as playing WebM locally often raises problems.

3.2 Data pre-processing

In this section, the pre-processing operations are covered. These include perturbed frame removal, smooth velocity visualization, and shortening of the bags.

3.2.1 Perturbed frame removal

To make the data more user-friendly, in addition to the raw data, our team provided a script that can be used to remove frames with a considerable amount of noise. Rosbags that were particularly noisy were processed using this script and will be provided alongside the original data.

An example of such data can be found in instances where the Go1 quadruped experiences heavy perturbations. Such instances can be identified by a large number of events occurring in a short amount of time. In practice, the fact was used that the DAVIS camera batches the events it observes every 30 ms, somewhat mimicking the workings of a traditional frame-based camera. These batches can each be inspected and filtered based on the number of events they contain. An example of such a frame can be seen in Fig. 6 and an example of a noisy frame can be seen in Fig. 7. These figures were generated using a script that our team developed. The positive events in these frames are white while the negative events are black. Pixels, where no events occurred in that frame, remain grey.

To achieve this, we propose a metric called the Pixel Variation Ratio, or PVR. The PVR represents the number of events relative to the total number of pixels in the DAVIS camera in percentages. The formula for calculating the PVR can be found in eq. (1).

$$\text{PVR} = \frac{\text{\# of events in a frame}}{\text{\# of pixels in a frame}} \times 100\% \tag{1}$$

The PVR values were computed for each frame and compared to a pre-defined threshold value. If a frame had a PVR value that crossed this threshold, this frame would be replaced by the most recent non-noisy frame. However, It is worth noting that some rosbags did not necessitate this process.

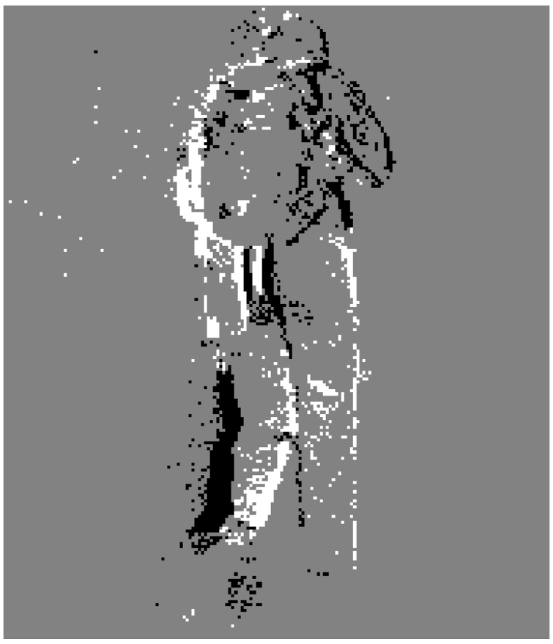


Figure 6: Example of normal event footage

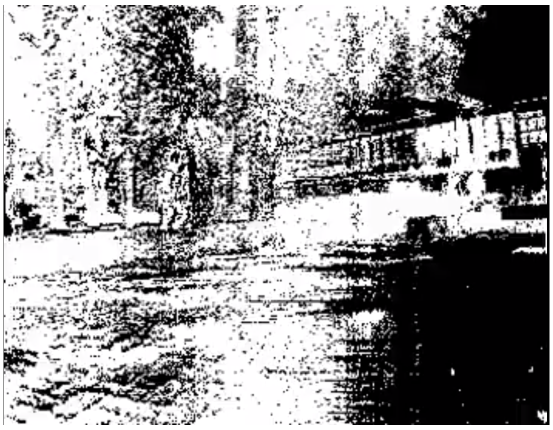


Figure 7: Example of noisy event footage

To gain an idea of the PVR distribution for each rosbag that was collected, Table 2 displays a five-number summary of the PVR values and whether a processed bag was provided alongside the original.

To pick the threshold consistently, a mix of trial and error with some exploratory data analysis was utilized. To that end, plots like the ones in Fig. 8 were used. Here, The boxplot and histogram use the PVR's of every frame in the relevant rosbag. For this bag, and other bags that needed processing, the percentage value of the third quartile was a good starting point. In the case of the Industrial Design Canteen bag, 60 percent was a good threshold value to use to remove noisy frames.

3.2.2 Smooth velocity visualisation

For the dataset to be usable, a groundtruth is needed. For this dataset, the provided groundtruth is the velocity measurement and IMU data. The velocity is measured by the robot in three directions and one rotation. This captured velocity data is very susceptible to minor movements of the robot. Whenever it takes a step, the body of the robot will

Scene	Required processing	Minimum	Q1	Median	Q3	Maximum
Outside IO	Yes	0.00	12.00	20.55	32.95	396.28
CoR hall	No	0.02	13.43	24.30	47.65	274.79
Bicycle rack	Yes	0.01	17.82	36.14	57.81	338.18
Grass	Yes	0.02	17.84	29.43	48.80	348.33
Industrial Design	Yes	0.14	4.50	22.78	49.98	255.60
Industrial Design Canteen	Yes	0.11	6.80	29.93	45.33	259.11
IO hall upstairs	No	0.01	0.02	6.49	20.02	98.59
Mirror	No	0.31	8.70	16.80	25.68	220.11
3mE Main Entrance	Yes	0.09	21.00	32.99	46.75	364.38
Basement hallway	No	0.09	6.50	10.98	16.61	180.23

Table 2: The 5 number summaries of each scene

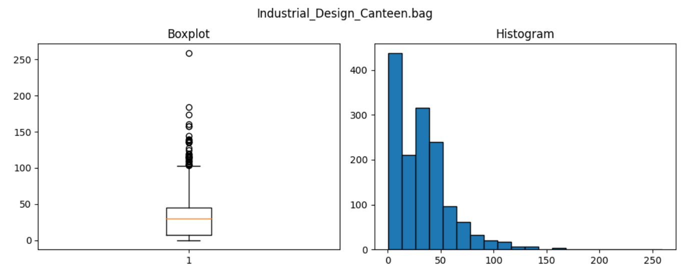


Figure 8: PVR data plots for the Industrial Design Canteen bag

vibrate. Next to this accurate, but rapidly vibrating data, a smoothed velocity curve is also provided, as can be seen in Fig. 9. This smoothed curve was extracted from the raw data by applying a Butterworth filter, which filters out the high-frequency oscillations. This smoothed curve might be less accurate to the sensor readings but is more representative of reality. With the provided raw data and code [9], used to smoothen the curve, a user can tweak the parameters to filter out more or less frequencies as desired.

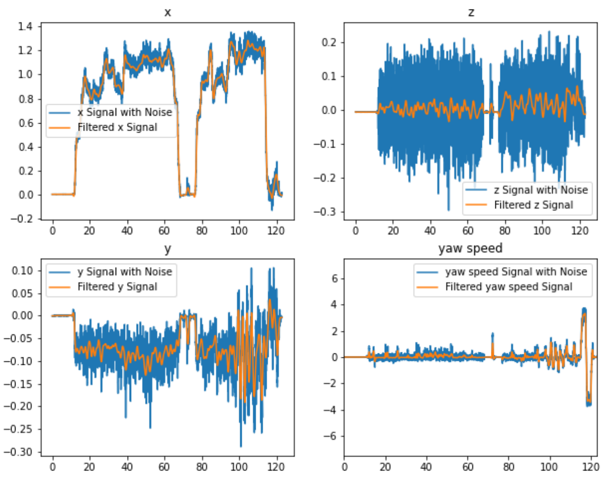


Figure 9: Smoothed velocity curve

3.2.3 Bag shortening

The LCM server, which provides the communication with the robot and the `unitree_legged_sdk`, has a tendency to randomly crash. As a result of such crashes, some recordings do not contain any new velocity measurements after the crash. This means that if the robot was walking forwards at a certain speed, the software will think it maintains that speed perfectly for the remaining duration of the dataset. This is showcased in Fig. 10. Without the correct velocity measurements, there is no groundtruth, which renders the part of the dataset after the crash unusable. Therefore, the rosbags are provided as one bag with the unusable part and a trimmed version cut right before the LCM crash. The cut is made at a quarter second before the crash, in this manner there is no incorrect data in the dataset, meaning that everything in the trimmed rosbag can be used.

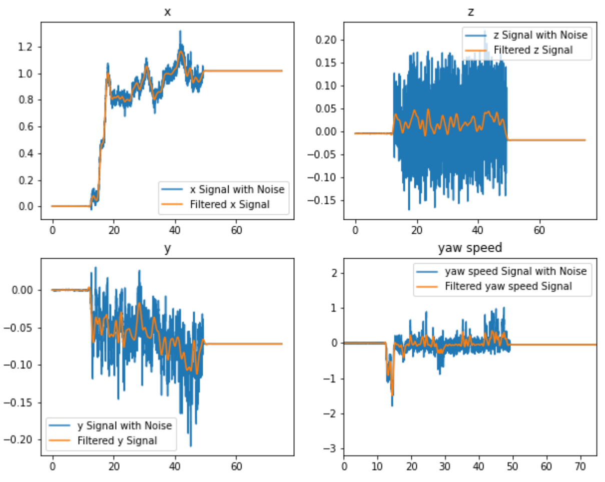


Figure 10: Industrial design raw and smoothed velocity data

4 Results

The scenes used for the dataset are summarized and shown in Table 3, and Fig. 11 respectively. The link to the entire dataset can be found in the repository [9]. The dataset needs to satisfy various scenarios for different training purposes, therefore a wide variety of complexity is present in the captured scenes. The most basic, baseline, scenarios captured are:

- "Basement hallway"
- "Outside IO"
- "Elevator mirror"
- "IO hall upstairs"

For medium-level complexity the following scenes were captured:

- "Industrial design"
- "Main entrance 3mE"
- "CoR hall"
- "Grass"

Lastly, the most complex scenes are:

- "Industrial design canteen"
- "Bicycle rack"

The first two scenes serve as baseline scenarios with minimal activity. To achieve this, footage was captured in a large indoor hall with a level, hard floor. One of these baseline scenarios was recorded indoors in the "basement hallway", while the other was captured outdoors in the "Outside IO" area.

Secondly, in addition to quiet indoor settings, three indoor scenes were recorded with varying levels of human presence to facilitate complicated scenario training. These scenes include the relatively quiet "Industrial design", the livelier "Main entrance 3mE", and "Industrial design canteen" with an abundance of people.

Thirdly, a scene with a wide range of distances, from close proximity to the camera to objects further away, was captured. For this purpose, the hallway in the CoR department of the 3mE building at the TU Delft was selected. This long hallway features tables on one side, providing objects at different distances. The presence of people in the hallway also allows for potentially complicated scenario training using the provided dataset. This scene is referred to as the "CoR hall".

For the fourth scenario, an outdoor environment with a soft floor was chosen to enable training of robots to walk on soft bedding. Outdoors on the grass, the high temporal resolution of the event-based camera can be effectively utilized by leveraging the shadows. This scene also facilitates the observation of objects in the far distance, such as cars or pedestrians. Additionally, the busy nature of the location offers more potential for complicated scenario training in outdoor environments. This scene is called "Grass".

The fifth scenario is designed to train robots for environments with mirrors. Footage of an elevator mirror in the 3mE building was captured for this purpose. This scene is referred to as the "Elevator mirror".

Finally, the "Bicycle rack" scene presents an indoor setting with a hard, level floor but with significant contrast and a large number of edges. This scene can be useful for training or testing edge detection algorithms.

An additional scene, "IO hall upstairs", has been included despite not fulfilling its initial purpose as a baseline scene. Although the unexpected appearance of people in the scene prevented its use as a baseline, the captured data can still provide valuable insights for other applications.

To gain a quick overview of the recorded data, a script was written in order to generate a timeline of the recorded sensor data. An example of such a plot can be seen in Fig. 12.

5 Discussion

In this section, potential improvements and flaws of our method that were encountered will be discussed. These include unexplored alternative methods and also external challenges, which left room for possible future improvements.

During the final phases of the project, `dv-ros`, a ROS driver made by IniVation themselves, was discovered [10]. This ROS package does not directly depend on `libcaer`, but on a more high-level interface. After a quick test with this driver, the results looked pretty similar to those of `rpg_dvs_ros`. However, future testing would be needed to clarify if there are any actual benefits of using this package instead of the `rpg_dvs_ros` package.

Another challenge that was encountered resulted from configuring the greyscale camera of the event-based camera to use auto-exposure, which allowed us to record datasets outside with greatly varying lighting conditions. However, this has probably resulted in the flickering of the intensity, and poor inside performance of this footage if any bright large windows are present. It could be beneficial for future work to recalibrate auto exposure for outside use and completely disable it when recording inside only datasets.

Another unexplored aspect is the configuration of the event threshold. For our datasets, it was chosen to leave it at the default settings. When setting this threshold lower or higher, the detail level can be in- or decreased. This is a trade-off which could differ for each use case. For future work, with a more specific application for the dataset in mind, this threshold can be configured to meet the needs of the application.

Another challenge we had to overcome was the instability of the LCM server. This limited the length of some datasets due to frequent crashes of the LCM server. Unfortunately, LCM had to be used because of the SDK version, which was determined by the locked firmware version for the other ongoing research projects. Newer versions of the `unitree_legged_sdk` do not use a LCM server for communication. It could be beneficial for future work to test if this new communication method is more stable. This also might allow the use of the `UnitreecameraSDK` to access the cameras [11], which makes the workaround using scraping obsolete.

To deal with noisy frames, the decision was made to remove them and replace them with the most recent non-noisy frame. While this works, it is one which gets rid

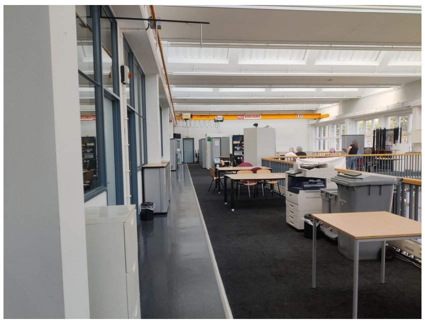
(a) Outside IO

(b) Basement hallway

(c) Industrial design

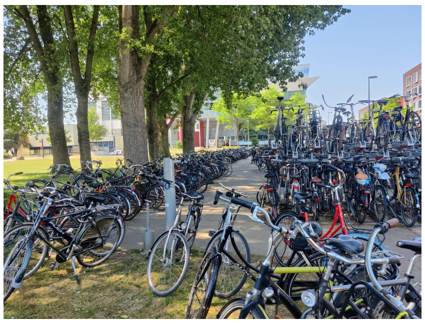
(d) Main entrance 3mE

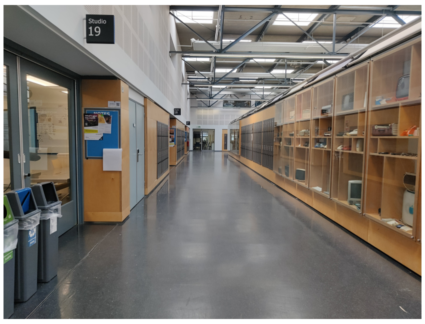
(e) Industrial design canteen


(f) CoR hall

(g) Grass

(h) Elevator mirror


(i) Bicycle rack



(j) IO hallway upstairs

Figure 11: Dataset scenes

Scene	Duration (s)	Benefit of this scene	Remarks
Outside IO	120	baseline, outside	-
Basement hallway	120	baseline, inside	-
Industrial design	49	quiet indoors	not full length
Main entrance 3mE	120	lively indoors	-
Industrial design canteen	57	crowded indoors	not full length
CoR hall	104	wide range of distance	not full length
Grass	120	soft floor, shadows for temporal resolution	-
Elevator mirror	41.5	mirror	not full length
Bicycle rack	86.5	large amount of edges	not full length
IO hall upstairs	35	bonus, indoor	not full length, unexpected appearance of people, not usable as baseline

Table 3: The recorded scenes, 'not full length' caused by LCM crash

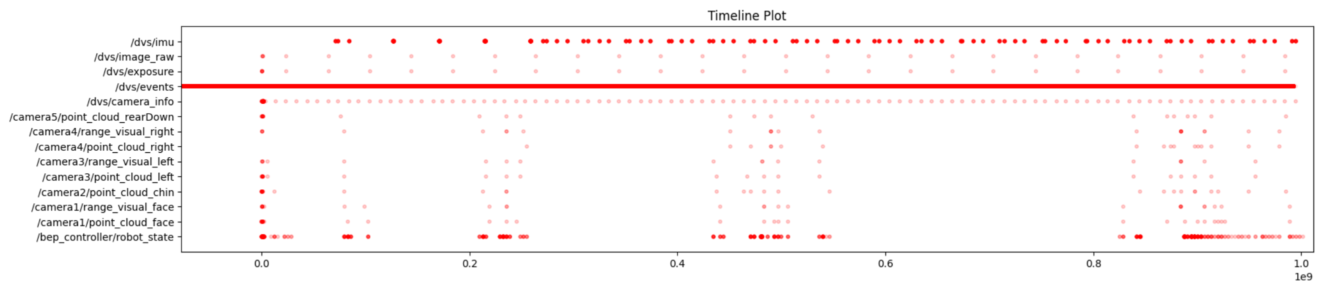


Figure 12: Example of timeline plot

of perhaps too much information, considering that not all information in a noisy frame is unrepresentative of reality. Moreover, the chosen approach has no regard for time in general, represented by jarring jumps in moments with consecutively noisy frames.

As such, two alternative approaches are recommended, namely; interpolating between the most recent and first-to-come non-noisy frames, in an attempt to capture the changes happening between frames, and culling only the noisy parts of noisy frames (parts with a disproportionate number of events).

One setback that was encountered when capturing the final dataset was the relatively large vibration and translation of the camera. This was caused by last-minute changes to the mounting system of the camera by the other project where the robot was employed. These changes moved the camera up by 7 cm which caused the vibrations in the dog to shake the camera much more than before. This has the undesired effect of some of the frames being covered almost entirely in events due to the sudden shock to the event-based camera. Our team supplied the parts to mount the backpack higher, and believe that with a good redesign of these parts the shaking could be decreased significantly. However, it is worth to note that the disturbance from the longer distance to the camera probably cannot be avoided entirely. For future work this could be reduced by further developing the pre-processing algorithm, or by mounting the camera on a more stable location on the robot.

A comparison can be made between our dataset and "*The Event-Camera Dataset and Simulator: Event-based Data for Pose Estimation, Visual Odometry, and SLAM*" [1]. Our dataset is slightly longer at 14.2 minutes instead of 8.7 minutes, which provides users with more data to train upon. This dataset also does not provide simulated scenes and motion tracking. The motion tracking provides very accurate ground truth, which is necessary when using only the IMU from the event-based camera as ground truth. However, such accurate tracking is not possible when using the robot dog outside of a lab. Instead, our dataset contains the robot dog's sensor data which can be used for ground truth. Another difference between the datasets is that their dataset uses an older model of the event-based camera with a lower resolution of 240 x 180 pixels.

Comparing to "*Biologically Inspired Mobile Robot Vision Localization*" [12], one feature that is lacking from our dataset, that theirs includes, is variation in time of recording. Ideally, future endeavours would capture scenes at different times of day. Event-based cameras function very well in poorly lit circumstances, whereas normal cameras tend to struggle in similar situations. Because of this, it is likely that robots that have to perform in poorly lit conditions will choose to use event-based cameras. Therefore, a database like ours, that does include different times of day could prove quite useful.

6 Conclusion and recommendation

Event-based cameras are the next big step in robot navigation as they offer low latency, high dynamic range and high temporal resolution. To be able to use this novel camera type for robot dog navigation, however, a full dataset from the perspective of a robot dog is needed to train the neural networks on. With the DAVIS346 event-based camera mounted on an Unitree Go1 robot dog, a full dataset with all of their sensors was collected. This includes events, grayscale images and IMU data from the camera. As well as 5 RGB cameras, IMU, point cloud, foot forces, joint angles, velocity and position data from the Go1. Data was collected at ten locations, both inside and outside, with a total (useable) length of almost 15 minutes. The ground truth can be found with the usage of the IMU and velocity data from the Go1. Furthermore, from the velocity data, a smooth curve was made in pre-processing. Another pre-processing step applied is the removal of 'bad' frames from the event-based camera, which are caused by the vibrating of the camera when the robot walks. The last pre-processing step applied was cutting the datasets to the correct length. Due to a crash that occurred, some datasets included incorrect data after a certain time, to ensure all data was factually correct and useable, the bags were shortened. After the pre-processing, 853 seconds of usable data including all the possible sensors from both the Go1 and the DAVIS346 remains.

Future endeavours could try to decrease the vibration in the camera, to ensure that fewer 'bad' frames are generated. Looking at a grander scale, however, future research could generate datasets mounted on different types of robots or possible cars. This hopefully makes event-based cameras more mainstream when it comes to navigation. To make event-based cameras truly viable, more research should also be done on the cameras themselves, which ideally, should increase the technologies accessibility significantly.

Acknowledgements

The completion of this research project would not have been possible without the contributions and support of a few individuals. We are deeply grateful to those who played a role in the success of this project.

Besides our supervisors, MSc. Chuhan Zhang, Dr. Jiatao Ding and Dr. Cosimo Della Santina, we would like to thank MSc. Vassil Atanassov for their invaluable input and knowledge throughout the research process. Their insights and expertise were instrumental in shaping the direction of this project.

References

[1] E. Mueggler, H. Rebecq, G. Gallego and T. Delbruck, 'The event-camera dataset and simulator: Event-based data for pose estimation, visual odometry, and slam,' *International Journal of Robotics Research*, vol. 16, no. 2, pp. 142–149, 2017.

[2] C. Brandli, R. Berner, M. Yang, S.-C. Liu and T. Delbruck, 'A 240 × 180 130 db 3 µs latency global shutter spatiotemporal vision sensor,' *IEEE Journal of Solid-State Circuits*, vol. 49, no. 10, pp. 2333–2341, 2014. DOI: 10.1109/JSSC.2014.2342715.

[3] E. Upton, *Supply chain, shortages, and our first-ever price increase*, 20th Oct. 2021. [Online]. Available: https://www.raspberrypi.com/news/supply-chain-shortages-and-our-first-ever-price-increase/ (visited on 13/06/2023).

[4] Unitree Robotics, *Unitreerobotics/unitree_legged_sdk at 3.4.2*, https://github.com/unitreerobotics/unitree_legged_sdk/tree/3.4.2, 2020. (visited on 13/06/2023).

[5] E. Mueggler, B. Huber, L. Longinotti and T. Delbruck, *Rpg_dvs_ros*, https://github.com/uzh-rpg/rpg_dvs_ros, 2020.

[6] Y. Hu and J. Roy, *Duguyue100/pyaer*, version 0.2.6, Mar. 2022. DOI: 10.5281/zenodo.6324301.

[7] Unitree Robotics, *Unitreerobotics/unitree_ros_to_real at v3.4.0*, https://github.com/unitreerobotics/unitree_ros_to_real/tree/v3.4.0, 2021. (visited on 13/06/2023).

[8] E. Olson, D. Moore and A. Huang, *LCM documentation*, Used Version 1.5.0, 19th Apr. 2023. [Online]. Available: https://lcm-proj.github.io/lcm (visited on 13/06/2023).

[9] J. van Brakel, D. van Loon, E. van Huffelen and R. Makinwa, *Bep-unitree-event*, https://github.com/SuperJappie08/BEP-Unitree-Event, 2023.

[10] iniVation, *IniVation AG / dv-core / dv-ros*, https://gitlab.com/inivation/dv/dv-ros, 3rd Mar. 2022. (visited on 14/06/2023).

[11] Unitree Robotics, *Unitreerobotics/UnitreecameraSDK*, https://github.com/unitreerobotics/UnitreecameraSDK/tree/ecd2058ba3f033af11c2d2a0306b44cc8d819332, 31st Dec. 2021. (visited on 15/06/2023).

[12] C. Siagian and L. Itti, 'Biologically inspired mobile robot vision localization,' *IEEE Transactions on Robotics*, vol. 25, no. 4, pp. 861–873, 2009. DOI: 10.1109/TRO.2009.2022424.