https://doi.org/10.2352/EI.2024.36.6.IRIACV-278

Real-time Cattle Intake Monitoring using Stereo Vision

Prajwal Rao[1], McKinley N Flinders[2], Dennis Buckmaster[3], Amy R Reibman[1], Jacquelyn P Boerman[2]
[1] Elmore Family School of Electrical and Computer Engineering, Purdue University, West Lafayette, U.S.
[2] Department of Animal Sciences, Purdue University, West Lafayette, U.S.
[3] Agriculture and Biological Engineering, Purdue University, West Lafayette, U.S.

Abstract
Accurate measurements of daily feed consumption for dairy cattle is an important metric for determining animal health and feed efficiency. Traditionally, manual measurements or average feed consumption for groups of animals have been employed which leads to human error and overall inconsistent measurements for the individual. Therefore, we developed a scalable non-invasive analytics system that leverages depth information derived from stereo cameras to consistently measure feed offered and report findings throughout the day. A top-down array of cameras faces the available feed, measures feed depth, projects depth to a 3-dimensional (3D) mesh, and finally estimates feed volume from the 3D projection. Our successful experiments at the Purdue University Dairy, that houses 230 cows, demonstrates its robustness and scalability for larger operations holding significant potential for optimizing feed management in dairy farms, thereby improving animal health and sustainability in the dairy industry. [1]

Introduction
Large cattle farms lack complete automation for monitoring tasks such as logging per-animal feed consumption. Cattle feed is composed of forages such as corn silage and hay crop silages, as well as concentrates like corn grain, soybeans, feed byproducts and minerals and vitamins which play a crucial role in improving daily gain, milk yield, and overall animal health. Additionally, varying composition of the feed mixture yield different textures and densities that affect traditional ways that feed is measured. Traditionally, feed is measured by manually weighing the total amount of feed offered (F_o) to a group of animals and the total amount of feed refused (F_r). The difference $F_o - F_r$ is total intake for the group, and the total intake is divided by the number of cows to obtain an estimate of individual cow intake. This lack of individual measurements provides inaccurate information for pinpointing specific health and efficiency metrics.

In this paper, we develop a robust analytics platform leveraging stereo cameras to process and obtain accurate feed volumes, and present them in a timely manner on a dashboard for farmers. Our completely automated system efficiently uses both the camera hardware and its host computer processing power.

Our system uses an OAK-D Power over Ethernet camera connected to a laptop using a network switch. The camera faces the feeding area covering two feed bunks in its field of view. From the camera, we extract the depth, convert it to a pointcloud representation, and then calculate the feed volume of the mesh generated from the pointcloud. Additionally, we provide a platform to store and visualize data for inference.

We have designed our system with the following goals:

- **Non invasive** - We estimate volume without disturbing the feed pile, which is achieved by having a top-down camera that views two feeding bunks simultaneously.
- **Reproducible, reliable and accurate** - We design a data collection component that enables us to reproduce existing volume estimates and enables us to explore the impact of algorithmic parameters. We accomplish this by saving only the information necessary to achieve these goals.
- **Real-Time** - Our pipeline is completely real-time. Processing, visualizing, and storing data is done in parallel leveraging multiprocessing queues.
- **Scalable** - Since one power-over-ethernet (POE) switch can drive multiple cameras, our system can process multiple feeding bunks at once.
- **Cost Effective** - Keeping low hardware cost in mind, one single host machine can drive multiple cameras using a single POE switch.

With our setup, we have collected 2 sets of 12 hours of data over two days, and we evaluate our estimates against real world measurements obtained by manually weighing feed. Our experiments demonstrate that our system is a viable approach to obtaining accurate estimates of available feed.

Related Work
Volume Estimation for Cattle Feed
Previous techniques [11] have introduced the use of machine vision with a near-infrared laser illuminator to detect a constellation of points that are then converted to a 3-dimensional representation. However, this technique relies on sweeping the camera over the feed pile that has been placed in a specialized enclosure to capture an array of points in its field of view. Then, the feed is made accessible to the cows. In another study, [4] focus on feed measurement over 5-second intervals by measuring the inter-frame depth difference between when the cow is eating (cow_{in}), and when it has stopped eating (cow_{out}). Feed is measured by calculating the difference between two cow_{out} frames over a region of interest.

In our paper, we also apply depth processing to estimate the volume of feed available. However, in our system, we directly mea-

[1]This research was supported by Purdue University's Colleges of Agriculture and Engineering Collaborative Projects Program 2022, and by the intramural research program of the U.S. Department of Agriculture, National Institute of Food and Agriculture, Inter-Disciplinary Engagement in Animal Systems, Grant # 2022-10737.

sure volumes at the feeding area rather than using a separate apparatus for feed measurement [11]. Additionally, we combine real-time depth estimation with a capture system and a visualization dashboard. This combination creates a powerful experimental framework that enables experiment construction and validation. In addition, our technique extends the depth sensing capabilities of the camera sensor by enabling real-time output while minimizing the computational load on the connected computer CPU.

Depth Processing using DepthAI

Existing work on feed estimation has demonstrated that using depth is a feasible approach to accomplish our task. We use DepthAI, a library provided by Luxinos (OAKD-PoE manufacturer) that provides useful APIs for depth estimation, object detection, and camera control. DepthAI uses nodes that individually provides a specific functionality as a building blocks. In particular, the stereo depth node calculates the disparity from the preprocessed left and right images of the camera, and routes it as a stream. We construct these node blocks for our use case, which is to compute disparity with filtering. To calculate disparity (defined as the distance in pixels between the pair of images for every pixel in the left and right pair), DepthAI captures images from the left and right cameras that form a stereo pair and matches each pixel of the pair using:

$$depth = \frac{focal\_length_{px} \times baseline}{disparity_{px}}. \tag{1}$$

Here the $focal\_length\_in\_pixels$ can be calculated from

$$focal\_length = \frac{width_{px} * 0.5}{\tan\left(hfov * 0.5 * \frac{\pi}{180}\right)}. \tag{2}$$

To obtain $hfov$ and $vfov$, we can use

$$hfov = 2 * \frac{180}{\pi} * \arctan\left(\frac{width * 0.5}{f_x}\right); \tag{3}$$

$$vfov = 2 * \frac{180}{\pi} * \arctan\left(\frac{height * 0.5}{f_y}\right). \tag{4}$$

The focal length (f_x and f_y) are provided by the intrinsic matrix (K) [13] of the camera.

$$K = \begin{pmatrix} f_x & s & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{5}$$

More information about calculating these values for our camera can be found in [3].

Real-Time Processing

Our use of queues and multiprocessing helps to obtain near-instant analysis of data, while making optimal use of the available resources of the camera and computer. Since DepthAI is designed to stream depth estimates from the camera image stream using queues, we incorporate the same principles for the rest of the system. To pass data between multiple processes through queues, we have to ensure that the data that we operate on are serializable [10]. In this work, we leverage a number of libraries for processing: DepthAI, Open3D, and Plotly. DepthAI is used to obtain depth data from the camera, Open3D for processing 3-dimensional (3D) data, and Plotly for visualization and interactive dashboards. However, some datatypes, such as meshes in Open3D, are not serializable, and therefore we process them in our main process and spawn separate processes for other operations. Additionally, every process runs at different speeds; for example, the camera outputs frames at a much faster rate than our main process which works on the data. Consequently, we grab only the latest frame from DepthAI, thus skipping older frames.

Method

System Components

To accomplish our aforementioned goals, our system is divided into four main components: Data Collection, Storage, Algorithm and Visualization. These are illustrated in Figure 1.

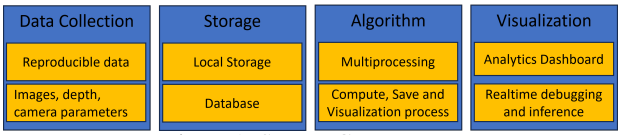


Figure 1: System Components

Data Collection

Our data collection component has been designed to serve two purposes: infer estimates from the available feed, and allow for the reproduction of computational experiments. It operates in tandem with other processes and can be configured to store any desired amount of data by modifying the configuration. More detail about this configuration can be found in our Configuration Section.

Storage

For storage, we use an external drive that can store more than a years' worth of data. Additionally, we have provided an option to store the data in the same schema to a MongoDB server [15]. With our experiments, we concluded that the speed of the system is unaffected by these storage choices.

Algorithm

The algorithm is designed to fulfill four main goals: speed, flexibility, extensibility, and concurrency. To achieve these goals, we leverage multiprocessing to create parallel processing of each code blocks, and FIFO (First In First Out) Queues to move data between each process. In our system, we have established three processes: Compute, Save, and Visualization. All the code is tested with python3.9 [8] but should operate with other versions as well.

Our volume computation system can be subdivided into two main blocks: the Capture process, and the Estimation process. To ensure continuous operation throughout the day, it is crucial to design an efficient system that is robust enough to operate continuously throughout a day. An overview of the algorithm is shown in Figure 2.

We found that it is not feasible to execute each block of Figure 2 serially because of the different processing speeds of each block. Therefore, we designed the system to execute some processes in

parallel. Specifically, we run our Compute process as the main process, and our Visualization and Save processes as child processes.

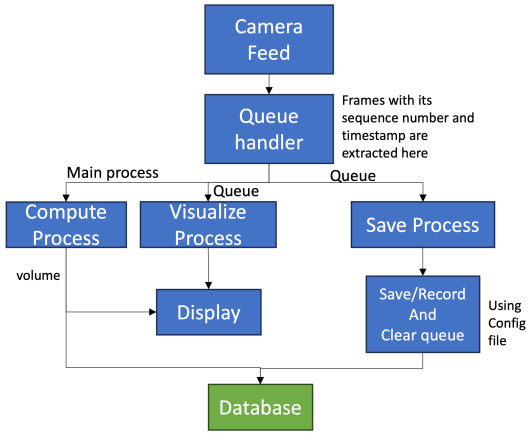


Figure 2: Overview of the Algorithm

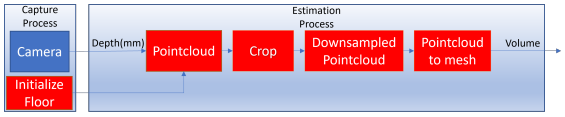


Figure 3: Compute process

Given depth data from the camera, our Compute process is designed to compute the corresponding volume estimates. We separate the Compute process into a Capture process and an Estimation process. The Capture process first captures the data from the camera; this includes the depth and color in a specific schema as shown in Schema 1. Then, our Estimation process computes the volume estimates with the depth information.

Compute Process

The Compute process is the main process which governs of all the processing required for depth-to-pointcloud, pointcloud preprocessing, depth-to-mesh, and mesh-to-volume computation. This process accepts the video feed from the queue handler as shown in Schema 1 and calculates its respective volumes.

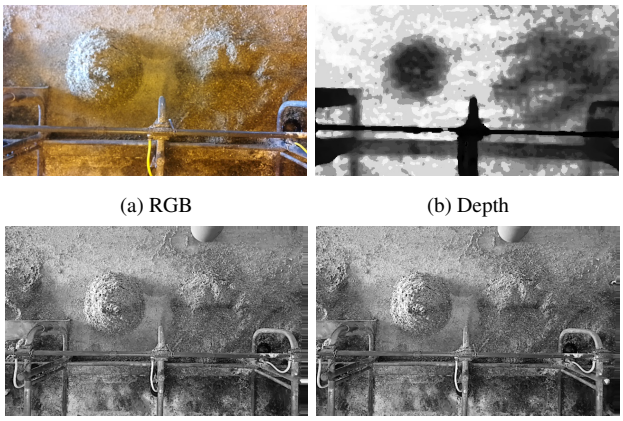
Capture process

We use the OAKD-PoE [7] camera for capturing video data. This camera is comprised of three sensors: Left (grayscale), Right (grayscale), and center (color). The left and right camera video stream is used to calculate our depth estimates and the color camera stream can be used to detect objects (person, cow, etc) which occlude the feed pile. Additionally, along with the camera array, the OAKD-PoE also has 2 LEON CPU cores for processing, an Image Signal Processor used for image processing, and 16 SHAVE cores for vector processing such as for neural network operations. More information about the camera hardware can be found in [5]. The camera is powered by a POE (Power-Over-Ethernet) switch which is in turn connected to the computer. Both power and data are transferred through an Ethernet port using an RJ45 Cable [16].

In the Capture process, before we start processing, we have to establish our floor distance (from the camera) either automatically or manually in our configuration. This step has to be done just once after the camera is set up. We provide three different options: RANSAC, manual, and camera API. RANSAC (Random sample consensus) is an automated method to estimate the floor distance. However, RANSAC requires the floor to appear in a significant region of the image to obtain accurate estimates. Manually, we can either set the floor in our configuration (by measuring the distance between the camera to the floor at the barn). Finally, it is possible to use the depth estimator from the camera API to estimate the distance to the floor, after we have chosen a region of the image that contains only the floor.

Queue handler

Our camera outputs a sequence of frames as separate streams for Right, Left and Center sensors depicted in Figure 4. These streams have to be synchronized based on their Message ID; this is handled by the queue handler. The queue handler receives information from each stream and constructs a message composed of a set of {`color, left, right, depth, msgID, timestamp`}. See Schema 1.



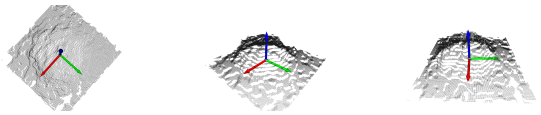
(a) RGB (b) Depth

(c) Left (d) Right

Figure 4: Sample image data from camera

```
{
  "msg": {
    "colorize": "color_img1",
    "rectified_left": "left_img1",
    "rectified_right": "right_img1",
    "depth": "depth_in_mm1",
    "sequence": "msg_sequence_number1",
    "time": "timestamp1"
  },
  "msg": {
    "colorize": "color_img2",
    "rectified_left": "left_img2",
    "rectified_right": "right_img2",
    "depth": "depth_in_mm2",
    "sequence": "msg_sequence_number2",
    "time": "timestamp2"
  },
}
```

Schema 1: Sample Data from camera

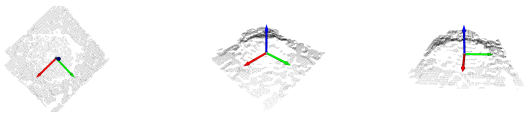


(a) Top View (b) Side View (c) Another View

Figure 5: Initial Pointcloud

Estimation process

An open-source library, Open3D [17], is used for 3D data processing such as pointclouds and meshes. We pass the depth data from the msg queue (Q_{msg}) depicted in Schema 1 into Open3D to obtain a pointcloud. A sample pointcloud of our feedpile is shown in Figure 5. Subsequently, this pointcloud data is then processed to extract only our target region (here the feed). Then, it is downsampled using voxel downsampling, and finally the outlier points are removed. With this preprocessed and cropped pointcloud, we set the (x,y,z) coordinate system to the center of the base of our cloud. While the base can also be predicted using RANSAC [12], here we manually set a global base value using the config file. This is to ensure that the pointcloud does not shift up and down due to the small changes in the prediction of the base at different instances. Now, our pointcloud represents the pile of feed as shown in Figure 6. Next, we convert the feed pointcloud to a Triangle mesh representation and calculate the floor points using the Delaunay triangulation [14] method.



(a) Top View (b) Side View (c) Another View

Figure 6: Processed Pointcloud

Finally, to measure the volume of the feed pile, we apply Equation (7) to sum the volume under every triangle (see Equation 6) of the mesh using simple 3D-geometry.

$$volume\_under\_triangle = (z_1+z_2+z_3)\,\frac{(x_1y_2-x_2y_1+x_2y_3-x_3y_2+x_3y_1-x_1y_3)}{6}. \quad (6)$$

Here (x_1,y_1,z_1), (x_2,y_2,z_2), (x_3,y_3,z_3) are the vertices of the triangle in Cartesian coordinates such that $z_1 \le z_2 \le z_3$. Therefore, for n triangles,

$$volume\_of\_mesh = \sum_{i=1}^{n} volume\_under\_triangle_i \quad (7)$$

```
{
  "sequence": "msg_sequence_number",
  "timestamp": "msg_timestamp",
  "depth": "depth_in_mm",
  "color": "color_img",
  "rectified_right": "right_img",
  "rectified_left": "left_img",
  "volume": "volume_meter_cubed"
}
```

Schema 2: Sample queue (Q_{show}, Q_{save}) data sent to Save, and Visualization processes

Finally, we create two queues - the show queue (Q_{show}) and the save queue (Q_{save}) from Q_{msg}. Each queue has a copy of the data along with the volume in cubic meters (m^3) which is then passed to the next two processes. A sample queue structure is depicted in Schema 2.

Visualization Process

A visualization dashboard is crucial for managing the data generated from the cattle barn environment as it provides a platform for real-time monitoring and data analysis of specific parameters. With a real-time dashboard, one can monitor changes in barn lighting throughout the day and also compare how these changes affect volume estimates by comparing camera frames at two time points. Additionally, overall failures in the system can be caught in an instant hence improving monitoring efficiency. Factors like lighting play a significant role in estimation accuracy. So by providing a clear, real-time overview of these variables, the dashboard allows us to make timely, informed decisions regarding deviations in feed volumes. Furthermore, the dashboard enhances operational efficiency by automating monitoring processes and can be configured to send alerts as and when needed.

The dashboard runs on the local network which can be accessed by phone or laptop on the same network. A sample visualization is shown in Figure 7. We present the volume vs time graph on the top left to view changes in volumes as the cow consumes feed. In addition, on the top right we have a 3D representation of the processed pointcloud at the selected timestamp, and at the bottom, we have the respective images from the camera - crop area, color, depth, left and right grayscale of the camera view.

Our visualization process is spawned using the inbuilt multiprocessing [6] library in python3.9 which runs in tandem with the estimation. We visualize the data from Q_{show} and categorically view color and grayscale (left and right) video using OpenCV [2] python library. For the depth, we first normalize the frame in the range $[0,255]$, perform histogram equalization, and add an colormap prior to visualization. Additionally, we have an on-click save operation which assists users to compare color, depth, and volume estimates between two or more timestamps.

Finally, we have designed our dashboard to be interactive. Hovering over timestamps in the volume vs time graph updates pointcloud of the region of interest, and their color, depth, left, and right frames.

Save Process

Unlike the visualization process, the save process is driven by Advance Python Scheduler [1], an external library written in python. The advantage of this scheduler is that it can be configured to run parts of python code at a pre-defined time instant. For example, a function can be run every 20th minute of the day, or every minute every alternate hour. This chronological job can be set using the configuration file. The saved data from Q_{save} can be stored by setting the folder location in the configuration file.

Configuration

We provide an external configuration file config.ini to control parameters for the Capture and Estimation process. Settings we consider are distance from the camera to the floor (`base_dist_real`),

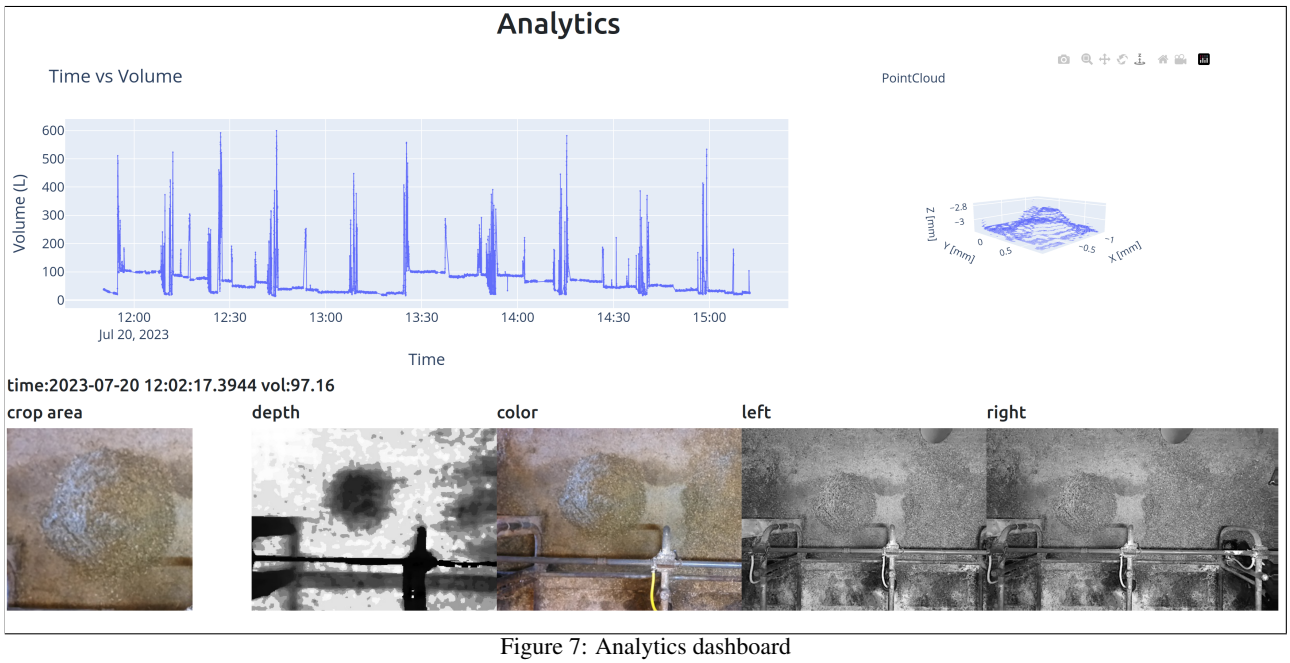


Figure 7: Analytics dashboard

pixel size of grayscale camera (`pixel_size`), pixel size of color camera (`pixel_size_color`), focal length (`focal_length`), crop bounds (`crop_min_bound`, `crop_max_bound`), output save location (`save_path`), number of frames to save (`save_range`), and saving frequency (`day`, `hour`, `minute`). For saving frequency, we use a cron-style configuration [9] to maintain platform consistency. A sample configuration is shown in Schema 3.

```
[DEFAULT]
record_time = 300
base_dist_real = 2.93
queue_clear_interval = 60

[depthai.config]
pixel_size = 3e-6
pixel_size_color = 1.55e-6
focal_length = 2.35e-3

[open3d.config]
crop_min_bound = [-0.2, -0.6, 2.0]
crop_max_bound = [0.6, 0.2, 3.0]

[save.config]
save_path = savedata/
save_range = 50
# no of minutes
day = *
hour = *
minute = *
```

Schema 3: Configuration File

Experiments

For our experimental setup, our goal was to measure volume given depth data collected from the camera. We also explore how lighting and shape of the feed pile affects our measurement. All data was gathered in the Purdue University Dairy Research Farm. Two diets of feed were used in the experiment: Ration-1 and Ration-2. Their densities were experimentally measured to be 0.325 Kg/L, and 0.323 Kg/L.

For each diet, we use five feed volumes (100, 80, 60, 40, 20 L). Table 1 shows the measured volumes and weights for the five different volumes of the two types of feed. Additionally, for our experiments, we also test two different lighting scenarios, and the presence and absence of a divot in the feed pile. The two lighting conditions are achieved by switching on and off a light mounted above the feed.

Our volume-estimation system is initiated to operate throughout the experiment. During the experiment, the feed is first manually measured with a scale that can measure up to 600 Kg with an error margin of ± 0.05 Kg. Then the weighed feed is manually placed on the floor of the feeding bunk where our system estimates the volume. Data is recorded and volume is estimated for each feed pile individually for a 5-minute duration. Thus, the experiment has 40 distinct conditions based on the arrangements of the feed pile: Ration-1 and Ration-2, 5 volumes, light on and light off, and without divot and with divot.

Table 1: Weight Samples Measured

Diet	Volume(L)	Density(Kg/L)	Weight(Kg)
Ration-1	100	0.323	32.3
	80	0.323	25.84
	60	0.323	19.38
	40	0.323	12.92
	20	0.323	6.46
Ration-2	100	0.35	35.0
	80	0.35	28.0
	60	0.35	21.0
	40	0.35	14.0
	20	0.35	7.0

Diet	Volume Ground Truth	Light and Divot conditions			
		Off/without	On/without	Off/with	On/with
Ration-1	100	100.0	97.7	88.5	82.9
	80	87.7	85.4	76.2	65.0
	60	71.4	65.7	47.0	45.6
	40	52.1	49.6	36.3	33.3
	20	32.4	25.5	27.3	21.5
Ration-2	100	113.4	98.8	99.9	96.1
	80	87.7	80.8	76.1	71.9
	60	69.2	63.08	49.7	44.9
	40	42.9	39.2	36.7	27.8
	20	29.0	26.4	23.9	25.7

Results

Our results are shown in Table 2. The two diets, Ration-1 and Ration-2, are weighed in subsets of 100, 80, 60, 40, and 20L each, as shown in the Volume Ground Truth column. The last four columns are the system estimates of volume with the different lighting and divot configurations. In these columns, the first set of two columns compare light off and on without a divot in the feed. The last two columns compare light off and on with a divot. Uncontrollable variables such as sunlight impact our estimates. For example, the 100L volumes differ between Ration-1 and Ration-2 when light is off and no divot. Additionally, when we compare with and without divot for one type of feed density, we observe that estimates are better for lower volumes as compared to higher volumes. This discrepancy is caused by two aspects of the visual appearance of larger piles. First, these piles tend to appear flatter due to the presence of the divot, and second, the system has difficulty accurately estimating volume at the center of the divot.

Conclusion

Our proposed system presents a cost-effective automatic strategy to measure volumes of cattle feed using a readily available camera system. Leveraging machine vision in commercial farms has significant potential for monitoring health, production, and efficiency. In this paper, we presented a non-invasive system that enables the monitoring of feed consumption in commercial farms using cost-effective hardware. To facilitate reproducibility, we developed a data collection component that securely stores the necessary observations. Additionally, we have created a interactive visualization dashboard for real-time monitoring and analysis.

We are continuing to evaluate our system at the Purdue University Dairy Research Farm. For our next steps, we will incorporate cow-detection, feeding pattern estimation, and identification into our architecture. Additionally, in our field trials, we have observed that measurements in an actual barn are challenging because of different lighting conditions throughout the day. Therefore, we will continue to explore the implications of lighting and camera configuration on the accuracy of our depth estimates.

References

[1] Apscheduler. `https://apscheduler.readthedocs.io/en/3.x/`.

[2] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[3] Disparity map. `https://docs.luxonis.com/projects/api/en/latest/components/nodes/stereo_depth/#calculate-depth-using-disparity-map`.

[4] J. Lassen, J.R. Thomasen, and S. Borchersen. Repeatabilities of individual measures of feed intake and body weight on in-house commercial dairy cattle using a 3-dimensional camera system. *Journal of Dairy Science*, 106(12):9105–9114, 2023.

[5] Luxinos hardware. `https://docs.luxonis.com/projects/hardware/en/latest/pages/rvc/rvc2/#hardware-blocks-and-accelerators`.

[6] Multiprocessing. `https://docs.python.org/3/library/multiprocessing.html`.

[7] OakD-POE. `https://shop.luxonis.com/products/oak-d-poe`.

[8] Python 3.9. `https://www.python.org/downloads/release/python-3913/`.

[9] Larry Reznick. Using cron and crontab. *Sys Admin*, 2(4):29–32, 1993.

[10] Serializable. `https://docs.python.org/3/library/pickle.html`.

[11] A.N. Shelley, D.L. Lau, A.E. Stone, and J.M. Bewley. Short communication: Measuring feed volume and weight by machine vision. *Journal of Dairy Science*, 99(1):386–391, 2016.

[12] Wikipedia contributors. Random sample consensus — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Random_sample_consensus&oldid=1182022297`, 2023. [Online; accessed 10-February-2024].

[13] Wikipedia contributors. Camera resectioning — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Camera_resectioning&oldid=1196552085`, 2024. [Online; accessed 10-February-2024].

[14] Wikipedia contributors. Delaunay triangulation — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Delaunay_triangulation&oldid=1198183130`, 2024. [Online; accessed 10-February-2024].

[15] Wikipedia contributors. Mongodb — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=MongoDB&oldid=1202299498`, 2024. [Online; accessed 12-February-2024].

[16] Wikipedia contributors. Registered jack — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Registered_jack&oldid=1195447078`, 2024. [Online; accessed 10-February-2024].

[17] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A modern library for 3D data processing. *arXiv:1801.09847*, 2018.