CHAPTER 1

Evaluation

In this chapter we devise a set of experiments to test the sensor system's properties and come to conclusions as to their effect on our ability to detect occupants. We then outline a process for taking raw sensor data and performing occupancy predictions with it. Using that process, we then devise a set of occupancy scenarios, and experiment with different machine learning algorithms abilities to determining occupancy from that data.

1.1 Sensor Properties

In order to best utilize the Melexis MLX90620 (*Melexis*), we must first understand the properties it exhibits, and their potential effects on our ability to perform occupancy measurements. These properties can be broadly separated into three different categories; bias, noise and sensitivity.

1.1.1 Bias

When detecting no infrared radiation (IR), the sensor should indicate a near-zero temperature, as the sensor's method of determining temperature involves measuring IR. If in such conditions the temperatures indicated are non-uniform, that suggests that the sensor has some level of bias in its measurements. We attempted to investigate the possibility of such bias by performing thermal captures of the night sky. While this does not completely eliminate the IR, it does remove a significant proportion of it.

To test this, the thermal sensor was exposed to the night sky at a capture rate of 1 Hz for 4 minutes, with the sensing results combined to create a set of means and standard deviations for the pixels at "rest". The average temperature detected was 11.78 °C, with the standard deviation remaining less than 0.51 °C over the entire exposure period. The resultant mean thermal map (Figure 1.1)

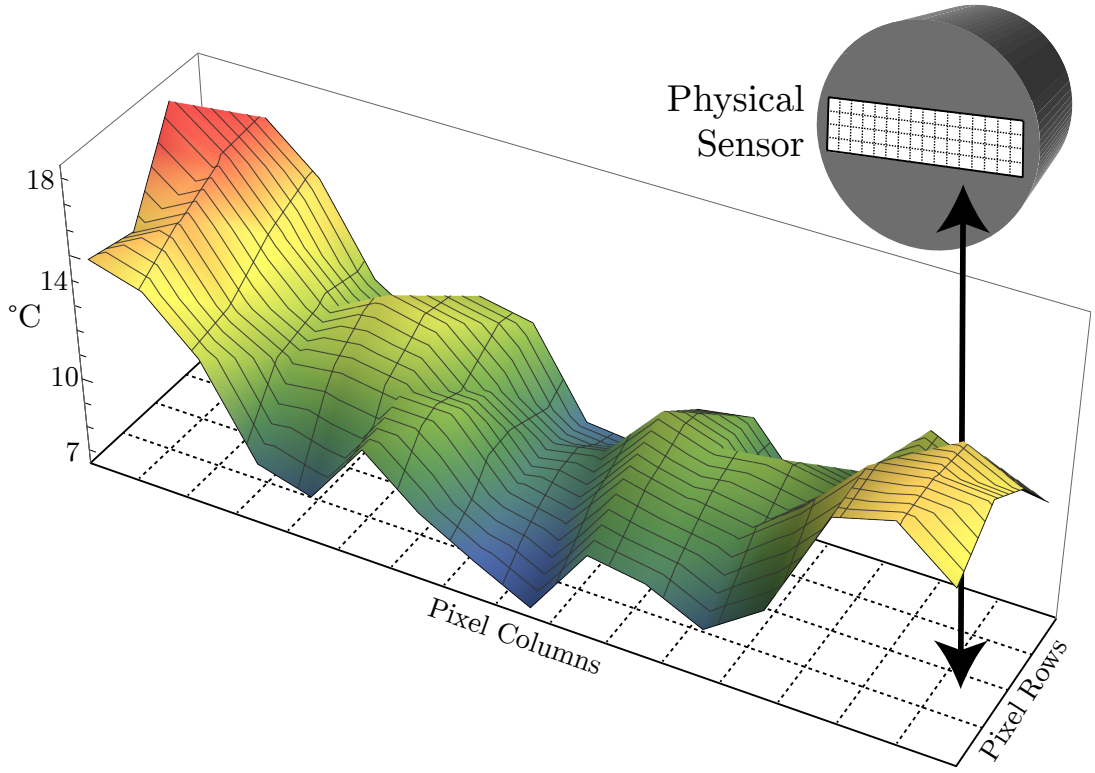

Figure 1.1: Mean values of 4 minute night sky thermal capture plotted over sensor's 16×4 grid

shows that the four center pixels maintain a similar temperature around 9 °C, with temperatures beginning to deviate as they became further from that point.

The most likely cause of bias is related to the physical structure of the sensor. The *Melexis* is a rectangular sensor which has been placed inside a circular tube. Due to this physical arrangement, the sides of this rectangular sensor will be significantly closer to these edges than the center. If the sensor's casing is at an ambient temperature higher than the measurement data (as they likely were in this case) thermal radiation from the sensor package itself could provide significant enough to cause the edges to appear warmer than the observed area of the sky. This effect could be controlled for by cooling the sensor package to below that of the ambient temperature being measured, however, this is not a concern in this project, as the method of calculating a thermal background will compensate for any such bias provided it remains relatively constant, which if our hypothesis is correct, it should.

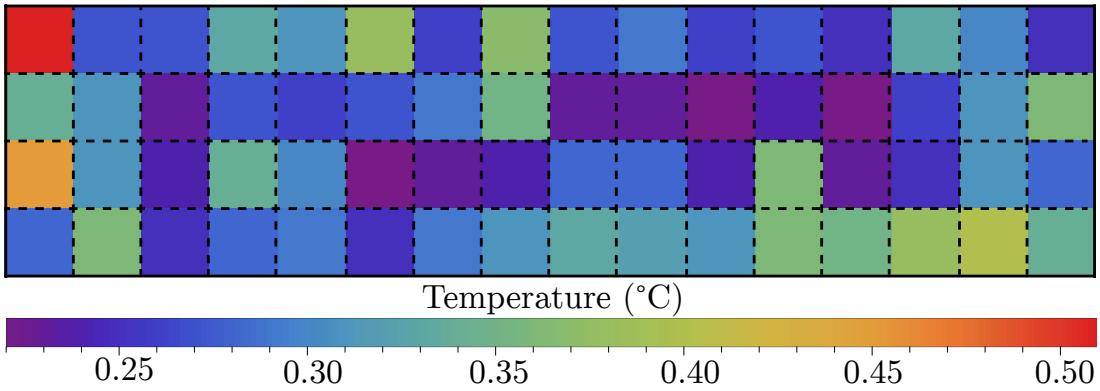

Figure 1.2: Standard deviation of 4 minute night sky thermal capture plotted over sensor's 16×4 grid

1.1.2 Noise

One of the primary features of the *Melexis* is the ability to sample the thermal data at a variety of sample rates between 0.5 Hz and 512 Hz. It was noted in preliminary experimentation that a higher sample rate appeared to result in noisier temperature readings. As our experiments focus on separating objects of interest from a thermal background, it is important to determine if this noise would post an issue for our use case.

Figure 1.3 plots one of the central pixels of the sensor in a scenario where it is detecting a background, and when it is viewing a person, at the five different sample rates achievable with the current hardware configuration. We can see in these plots that the data becomes significantly more noisy as the sample rate increases, and we can also conclude that the sensor uses a form of data smoothing at lower sample rates, as the variance in data increases with sample rate. If the sample rate were to increase beyond a certain threshold, it is likely that the ability for the sensing system to disambiguate between objects of interest and the background would diminish.

In the 0.5 Hz case, the third standard deviation above background ($3\sigma_b$) is 6.4 °C below the minimum occupant value ($\min(o)$) detected. As the noise increases, this gap slowly decreases, with 5.75 °C for 1 Hz, 5.53 °C for 2 Hz, 4.48 °C for 4 Hz, and finally 3.15 °C for 8 Hz. In none of these cases is $3\sigma_b \geq \min(o)$, which would completely rule that sample rate out.

Based on the data, noise will not pose any issue as the slowest sampling rate of 0.5 Hz is sufficient for the system's needs, and shows a sufficiently large gap between occupant and background temperatures. Higher sample rates are unnecessary as occupancy estimations at a sub-second level present little additional

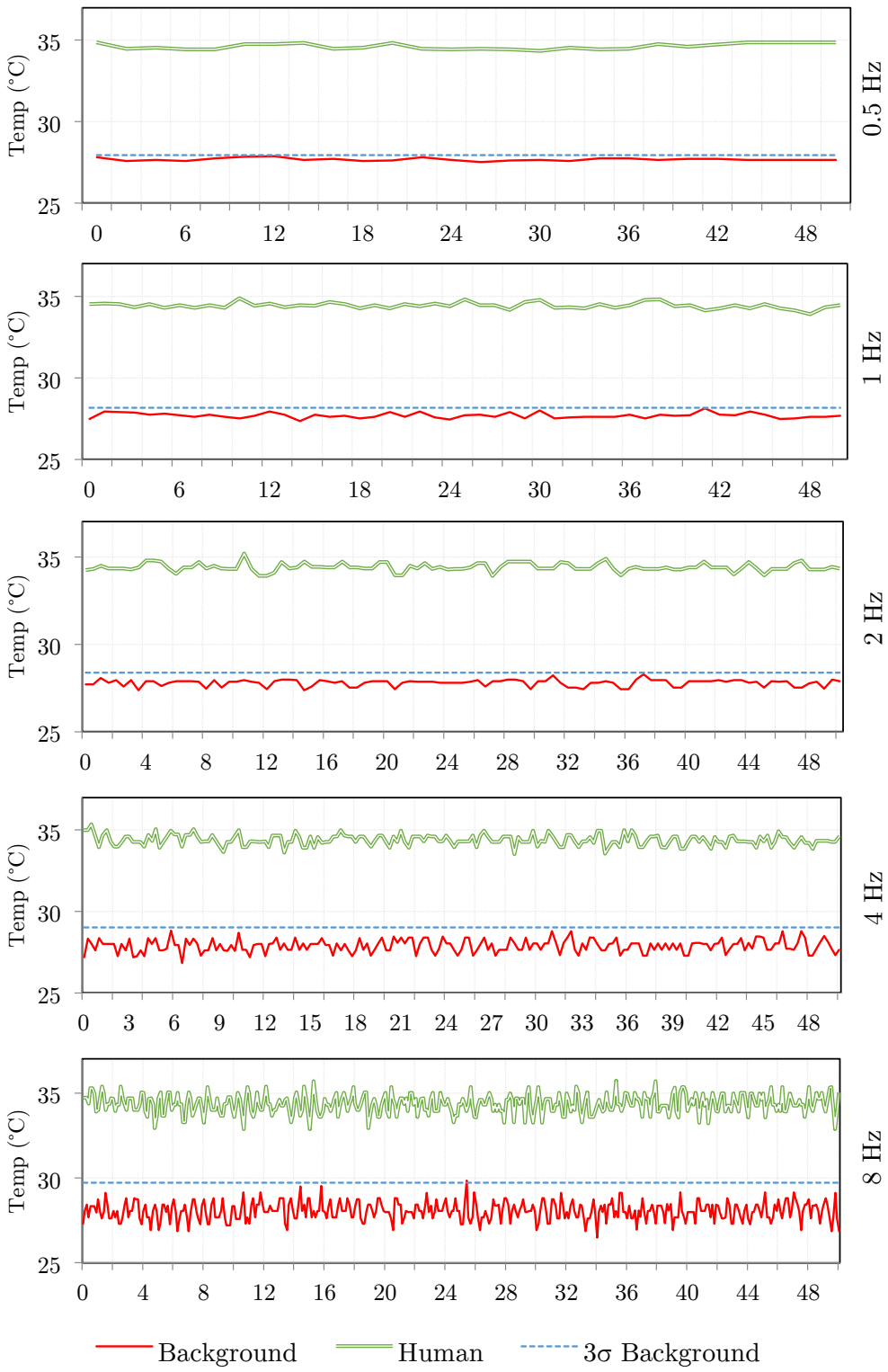

Figure 1.3: Plot of occupant and background sensor noise at sampling speeds 0.5 Hz – 8 Hz

value.

1.1.3 Sensitivity

The *Melexis* is a sensor composed of 64 independent non-contact digital thermopiles, which measure infrared radiation to determine the temperature of objects. While they are bundled in one package, the sensor's block diagram discussed previously (Figure **??**) shows that they are in fact wholly independent sensors placed in a grid structure. This has important effects on the properties of the data that the *Melexis* produces.

Figure 1.4 shows a smoothed temperatures graph of six of the sensor's central pixels as a hot object is moved from left to right at an approximately constant speed. One of the most interesting phenomena in this graph is the variability of the object's detected temperature as it moves "between" two different pixels; there is a noticeable drop in the objects detected temperature. Each pixel appears to exhibit a bell-curve like line, with the detected temperature increasing from the baseline and peaking as the object enters the center of the pixel, and the detected temperature similarly decreasing as the object leaves the center.

This phenomenon has several possible causes. One likely explanation is that each individual pixel detects objects radiating at larger angles of incidence to be colder than they actually are: As the object enters a pixel's effective field of view, it will radiate into the pixel at an angle that is at the edge of the pixel's ability to sense, with this angle slowing decreasing until the hot object is directly radiating into the pixel's sensor, causing a peak in the temperature reading. As the object leaves the individual element's field of view, the same happens in reverse.

This phenomena is not anticipated to impact our intended use case, as it only strongly affects objects at pixel or sub-pixel size. In our experimental conditions the sensor will not be sufficiently distant that humans could be detected as such sizes.

1.2 Classification

With a greater understanding of the prototype's caveats, it is now possible to gather both thermal and visual data in a synchronized format. This data can be collected and used to determine the effectiveness of the machine learning classification algorithms used. Due to the prototype's technical similarly to ThermoSense [1], a similar set of experimental conditions will be used, with a comparison against ThermoSense being used as a benchmark. To this end, several

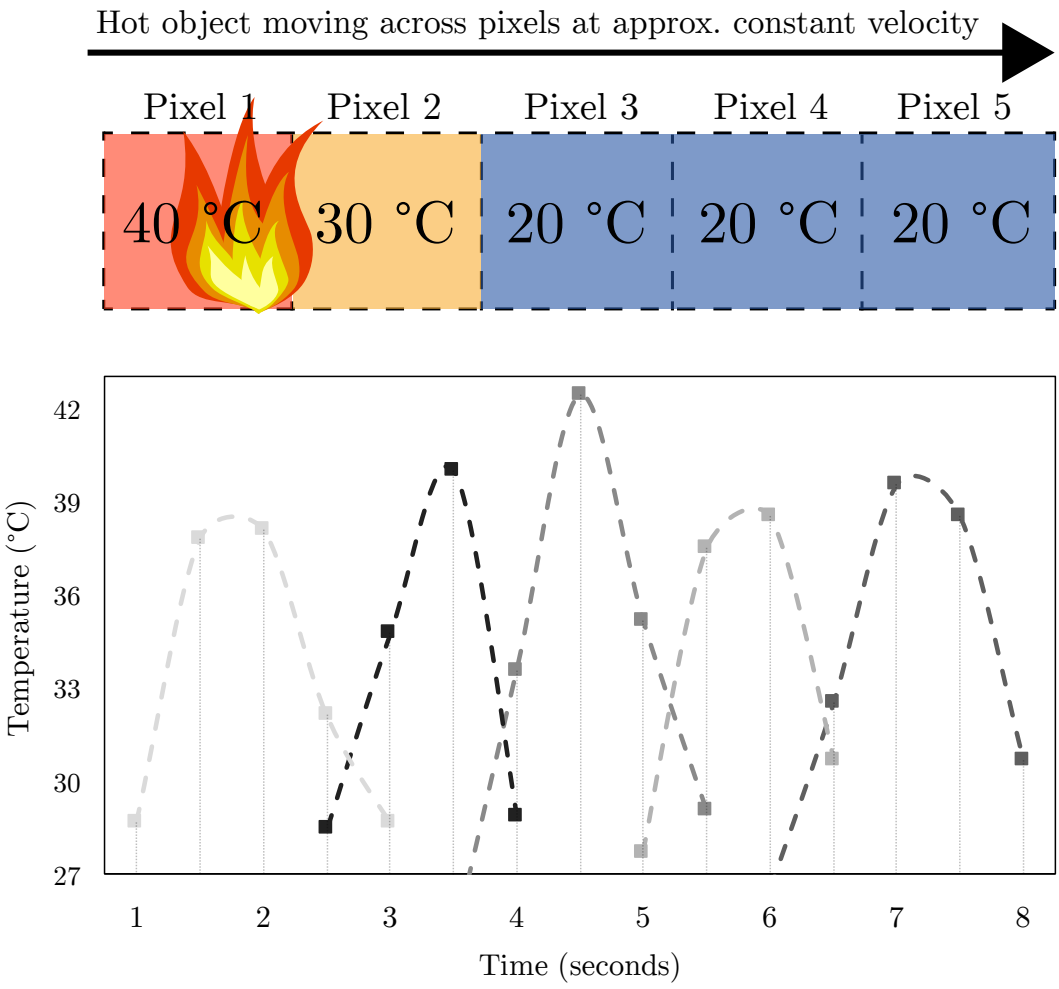

Figure 1.4: Temperature plot of five of the MLX90620's pixels as a hot object moves across them at a constant velocity

experiments were devised, each of which had its data gathered and processed in accordance with the same general process, outlined in Figure 1.5 and discussed in more detail in this section.

1.2.1 Data gathering

As the camera and the Arduino are directly plugged into the Raspberry Pi, all data capture is performed on-board through SSH, with the data being then copied off the Pi for later processing. To perform this capture, the main script used is `cap_pi_synced.py`.

`cap_pi_synced.py` takes two parameters on the command line; the name of the capture output, and the number of seconds to capture. The script initializes the `picamera` library, then passes a reference to it to the `capture_synced` function within the `Visualizer` class. The class will then handle sending commands to the Arduino to capture data in concert with taking still frames with the Raspberry Pi's camera.

When the script runs, it creates a folder with the name specified, storing both the thermal capture inside a file named `output_thermal.hcap`, and a sequence of files with the format `video-%09d.jpg` corresponding to each visual capture frame.

1.2.2 Data labeling

Once this data capture is complete, the data is copied to a computer with GUI support for labeling. The utility `tagging.py` is used for this stage. This script is passed the path to the capture directory, and the number of frames at the beginning of the capture that are guaranteed to contain no motion. This utility will display frame by frame each visual and thermal capture together, as well as the computed feature vectors (based on a background map created from the first n frames without motion).

The user is then required to press one of the number keys on their keyboard to indicate the number of people present in this frame. This number will be recorded in a file called `truth` in the capture's directory. The next frame will then be displayed, and the process continues. This utility enables the quick input of the ground truth of each capture, which is necessary for the classification stage.

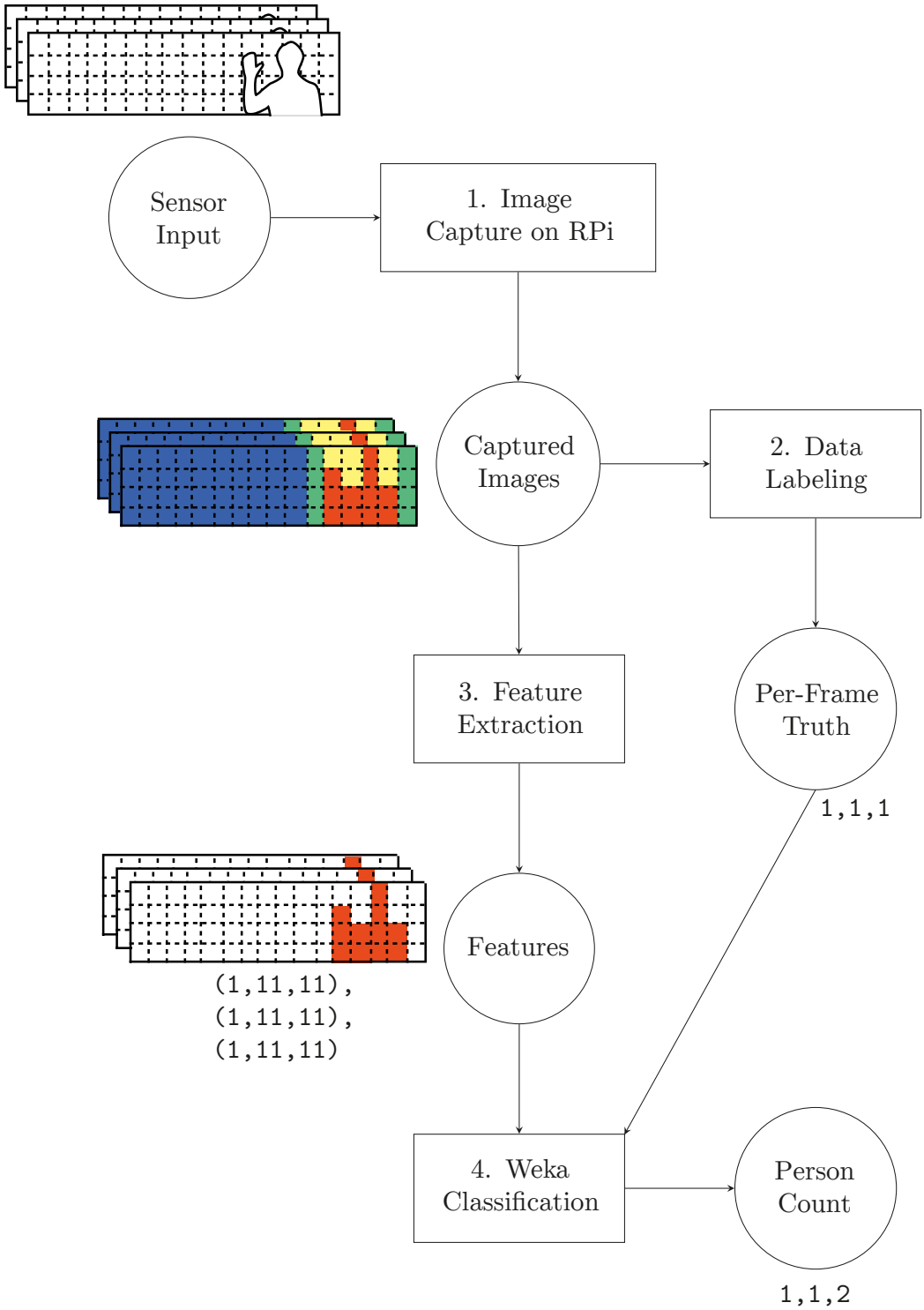

Figure 1.5: Process flow diagram for turning raw sensor input into occupancy estimates

1.2.3 Feature extraction and data conversion

Once the ground truth data is available, it is now possible to utilize the data to perform various classification tests. For this, we use version 3.7.12 of the open-source Weka toolkit [2], which provides easy access to a variety of machine learning algorithms and the tools necessary to analyze their effectiveness.

We collate and export the ground truth and extracted features from multiple captures to a Comma Seperated Value (CSV) file for processing with Weka. `weka_export.py` takes two parameters, a comma-separated list of different experiment directories to pull ground truth and feature data from, and the number of frames at the beginning of each capture that can be considered as "motionless." With this information, a CSV-file file is generated.

1.2.4 Running Weka Tests

Once the CSV file is generated, it is then possible process this file through Weka. Weka provides a variety of machine learning classification algorithms to choose from. To investigate ThermoSense's results, we replicate their algorithmic implementations in Weka. We discuss the operation of each of these algorithms generally in Chapter **??**. In total, Thermosense uses three machine learning classifiers:

Firstly, they use a neural network with a hidden layer of five neurons, with a sigmoid activation function for the hidden layer and a linear activation function for the output layer. They test only the one, two and three person cases, relying on their Passive Infrared Sensor (PIR) to detect the zero person case. They use 70% of their data for training the neural net, 15% for testing the net and the final 15% for validating their results. ThermoSense conducts tests interpreting the number of people as a numeric attribute.

We use Weka's "MultilayerPerceptron" neural network to recreate this, which creates a hidden layer of $(attributes + classes)/2$ (three) by default, however we manually reconfigure this to be one hidden layer of five neurons, like ThermoSense. It uses a sigmoid activation function for all neurons, except in the case that a numerical answer is to be predicted, in which case like ThermoSense, it uses a linear activation function for the output layer.

Secondly, they use a k-Nearest Neighbors (where $k = 5$) with the distance between features being determined by their Euclidean distance. For determining the class label, higher weightings are given to training points inversely to their distance from the point being classified.

Weka's "iBk" function is used to perform a KNN calculation, configuring `distanceWeighting` to be "Weight by 1-distance" and `KNN` to be 5, to make the classification as similar in function to the ThermoSense approach as is possible. ThermoSense does not specify what validation technique they used, so we elected to use a 10-fold cross-validation.

Thirdly, they use a Linear Regression model of $y = \beta_A A + \beta_S S + \beta$, whereby A is the number of active pixels, S is the size of the largest connected component, and the β values represent the corresponding coefficients. They opt to exclude the third feature, the number of connected components, as their testing indicates that excluding it minimizes the Root Mean Squared Error (RMSE) further.

We use Weka's "LinearRegression" function, and exclude the `numconnected` attribute from the feature vector list, as ThermoSense does, to attempt to match this approach.

In addition to the above three techniques, we also use nominal versions, in addition to our own techniques (detailed in). Our techniques are predominantly well known, and as with the ThermoSense techniques above, are described in more detail in Chapter **??**. The one algorithm that isn't particularly well known that we have chosen is the K* algorithm, which we describe in further detail here:

The KStar (K*) algorithm, developed by Cleary and Trigg [**?**] presents a different approach to k-nearest Neighbors type algorithms. With K* the distance used to compare similar points is not the Euclidean distance, but rather an entropic distance, a measure of how much effort is required to convert one example into another. This has several positive effects; it makes the algorithm more robust to missing values, and also it enables the classifier to output either a numeric or nominal result.

We have decided to use K* as one of our classification algorithms as it presents an interesting and different approach to the more well known algorithms above, and also allows the investigation of KNN-like techniques in the numeric area. K* is present in Weka as "KStar," and we will opt to use it in its default state.

To help maximize the efficiency of the classification tasks, we use the Weka's Knowledge Flow interface, which provides a drag-and-drop method of creating complex input and output schemes involving multiple different classification algorithms. We generate an encompassing flow that accepts an input CSV file of the raw data, and performs all numeric and nominal classification at once, returning a text file with the results of each of the different classification techniques run. The Knowledge Flow's structure can be seen in Chapter **??**. To further automate this, a Jython script, `run_flow.py` is used to call the flow through Weka's Java API. After this is complete, the script then runs a series of regular expressions on

the output text data to generate a summary CSV file with the relevant statistical values.

For those tests that are "nominal," the `npeople` attribute was interpreted as nominal using the "NumericToNominal" filter, which creates a class for each value detected in `npeople`'s columns. For those tests that are "numeric," `npeople` is left unchanged, as by default all CSV import attributes are interpreted as numeric. For all tests where not specifically stated otherwise, we use 10-fold cross-validation to validate our results.

As the data we are using is based on real experiments, the number of frames which are classified as each class may be unbalanced, which could in turn cause a bias in the classification results. We found that in most cases, the data that most unbalanced the set was that of the zero case, as it was the case most present in the data. As ThermoSense previously demonstrated, the use of the PIR alone allows for determining the zero/not-zero case effectively without classification algorithms. Due to this, we attempt to rebalance our dataset by excluding all zero cases from the data Weka receives.

1.2.5 Classifier Experiment Set

In our first set of experiments, a scene was devised in accordance with Figure 1.6 that attempted to sense people from above, as did ThermoSense. The prototype was set up on the ceiling, pointing down at a slight angle. For ease of use, the prototype was powered by mains power, and was networked with a laptop for command input and data collection via Ethernet. This set of experiments involved between one and three people being present in the scene, moving in and out in various ways in accordance with the following scripts.

The first four sub-experiments involved people standing;

- Sub-exp 1: One person walks in, stands in center, walks out of frame.
- Sub-exp 2: One person walks in, joined by another person, both stand there, one leaves, then another leaves.
- Sub-exp 3: One person walks in, joined by one, joined by another, all three stand there, one leaves, then another, then another.
- Sub-exp 4: Two people walk in simultaneously, both stand there, both leave simultaneously.

The latter five sub-experiments involved people sitting;

- Sub-exp 5: One person walks in, sits in center, moves to right, walks out of frame.
- Sub-exp 6: One person walks in, joined by another person, both sit there, they stand and switch chairs, one leaves, then another leaves.
- Sub-exp 7, 8: One person walks in, joined by one, joined by another, they all sit there, one leaves, one shuffles position, then another leaves, then another. (x2)
- Sub-exp 9: Two people walk in, both sit there simultaneously, both leave simultaneously.

In these experiments people moved slowly and deliberately, making sure there were large pauses between changes of action. The people involved were of average height, wearing various clothing. The room was cooled to 18 degrees for these experiments.

Each experiment was recorded with a thermal-visual synchronization at 1 Hz over approximately 60-120 second intervals. Each experiment had 10-15 frames at the beginning where nothing was within the view of the sensor to allow the thermal background to be calculated. Each frame generated from these experiments was manually tagged with the ground truth value of its number of occupants using the tagging script discussed in Subsection 1.2.2.

The resulting features and ground truth were combined and exported to CSV allowing Weka to analyze them. This data was analyzed with the feature vectors always being considered numeric data and with the ground truth considered both numeric and nominal. We evaluate the dataset against a set of classification algorithms introduced in Section **??** of our Literature Review.

Type	**Attribute**	**Weka Class & Parameters**
Neural Network (ANN)	Nominal, Numeric	`weka.classifiers.functions .MultilayerPerceptron -L 0.3 -M 0.2 -N 500 -V 15 -S 0 -E 20 -H 5`
k-nearest Neighbors (KNN)	Nominal, Numeric	`weka.classifiers.lazy.IBk -K 5 -W 0 -F -A "weka.core.neighboursearch .LinearNNSearch -A \"weka.core .EuclideanDistance -R first-last\""`
Naive Bayes	Nominal	`weka.classifiers.bayes.NaiveBayes`
Support Vector Machine (SVM)	Nominal	`weka.classifiers.functions.SMO -C 1.0 -L 0.001 -P 1.0E-12 -N 0 -V -1 -W 1 -K "weka.classifiers.functions .supportVector.PolyKernel -C 250007 -E 1.0"`
C4.5 Decision Tree	Nominal	`weka.classifiers.trees.J48 -C 0.25 -M 2`
K*	Nominal, Numeric	`weka.classifiers.lazy.KStar -B 20 -M a`
Linear Regression	Numeric	`weka.classifiers.functions .LinearRegression -S 0 -R 1.0E-8`
0-R	Nominal, Numeric	`weka.classifiers.rules.ZeroR`

Table 1.1: Weka parameters used for different classifications algorithms

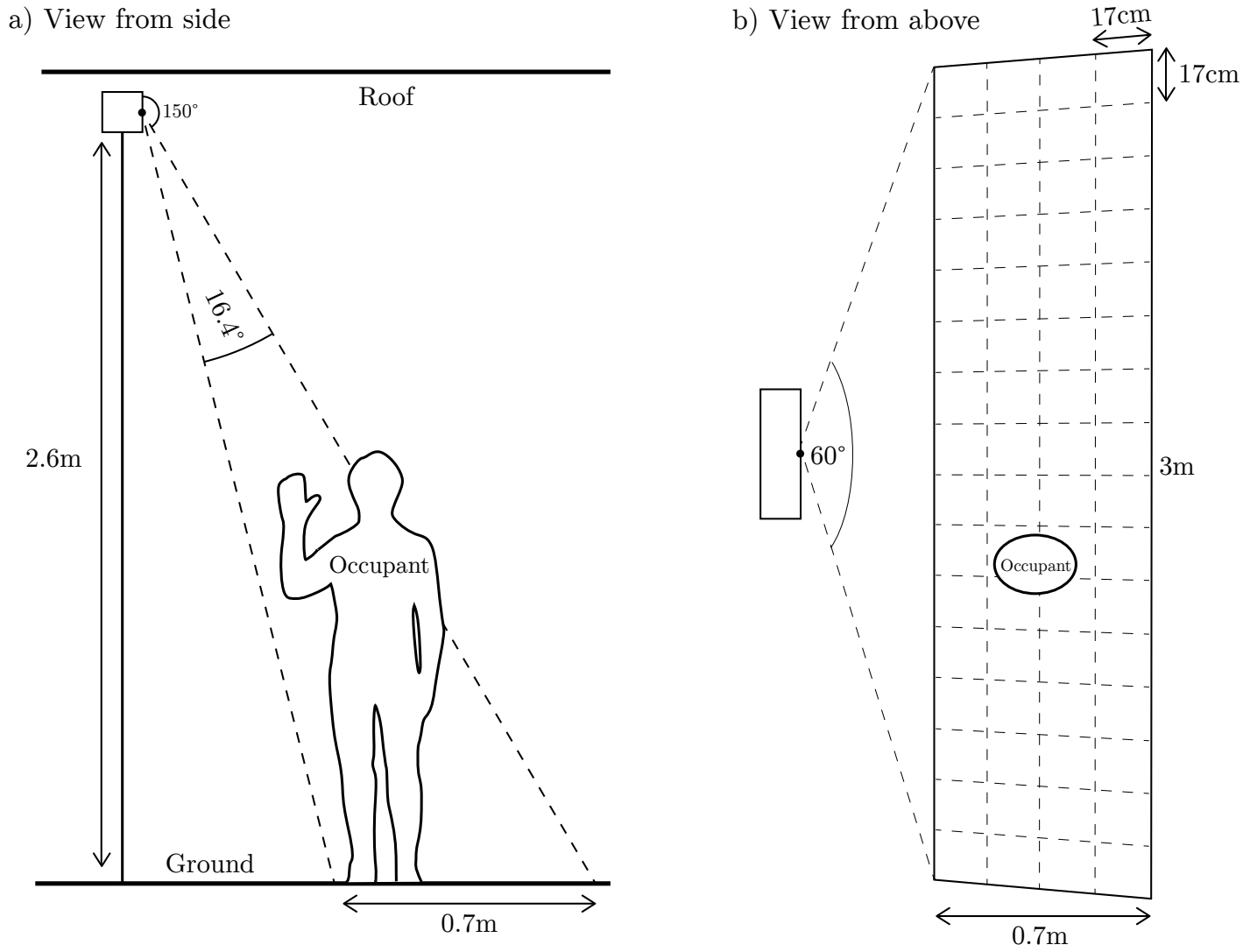

Figure 1.6: Classifier Experiment Set Setup (measurements approximate)

1.3 Results

1.3.1 Classification

Our results (Table 1.2) show an interesting spread of accuracies between the different tests that were tried. We will analyze the data with reference to two broad categories; those tests replicating ThermoSense, and those tests we ourselves proposed.

As discussed previously, significant care was taken to ensure that the same classification parameters were used between our experiments and those performed in ThermoSense to provide as accurate as possible a comparison between our results. However, there were some ambiguities with the ThermoSense results that have made it more difficult to determine which parameters to choose. In particular, with reference to the k-Nearest Neighbours tests (KNN), it was ambiguous within the ThermoSense paper as to if they had elected to use a nominal classification or a numeric classification for this data.

Because of this, four tests were performed overall to replicate the ThermoSense results as closely as possible; KNN tests for both numeric and nominal representations of data, a Multi-Layer Perceptron numeric test (MLP) and a Linear Regression numeric test (Lin Reg). With these tests we found that our prototype did not achieve comparable results. ThermoSense reported correlation coefficients (r) of around 0.9 for their MLP and Lin Reg tests, however we could not replicate these results, with our best being 0.69 and 0.59 respectively. We were also unable to achieve the low Root Mean Squared Errors (RMSEs) reported by ThermoSense, with their RMSEs for KNN, MLP and Lin Reg being 0.346, 0.385 and 0.409 respectively, while ours were 0.364 (KNN Nominal Case), 1.123 (KNN Numeric Case), 0.592 (MLP) and 0.525 (Lin Reg). Our numeric KNN test performed worse than the 0-R benchmark for numeric tests, indicating a very poor result, with it achieving an RMSE of 1.123 vs. the 0-R's 0.651.

For our own proposed nominal classification algorithms, our accuracies were significantly improved, and in some cases exceeded the RMSEs reported by ThermoSense. Within our dataset, the K* and C4.5 algorithms were most accurate, with accuracies of 82.56% and 82.39% respectively. They both achieved RMSEs lower than the best achieved by ThermoSense, with their 0.304 and 0.314 a significant improvement on ThermoSense's KNN RMSE of 0.346.

Following down the ranking, our nominal MLP performed next best, with an accuracy of 77.14%, and an RMSE of 0.362, which is slightly higher than ThermoSense's best result. Following, the Support Vector Machine (SVM) implementation achieved a relatively poor accuracy of 67.18% with an RMSE of

Classifier	RMSE	Precision (%)	Correlation (r)
ThermoSense Actual			
KNN[1]	0.346		
Lin Reg	0.385		0.926
MLP	0.409		0.945
ThermoSense Replication			
KNN[1]	0.364	65.65	
MLP	0.592		0.687
Lin Reg	0.525		0.589
KNN[1]	1.123		0.377
Numeric			
K*	0.423		0.760
0-R	0.651		-0.118
Nominal			
K*	0.304	82.56	
C4.5	0.314	82.39	
MLP	0.362	77.14	
SVM	0.398	67.18	
N. Bayes	0.405	63.59	
0-R	0.442	49.74	

[1]: Included zero in training data
%: Precision, for a nominal test
r: Correlation coefficient, for a numeric test

Table 1.2: Classifier Experiment Set Results

0.398, and finally the Naive Bayes (N. Bayes) approach, achieved the worst accuracy of 63.59% with an RMSE of 0.405. None of these techniques however achieved an RMSE or accuracy worse than our 0-R benchmark, which achieved an RMSE of 0.442 and an accuracy of 49.74%.

In our sole numeric choice of K*, we found that it achieved a better correlation than any ThermoSense technique, with $r = 0.760$. Additionally, its RMSE of 0.423 was also superior.

1.3.2 Energy Efficiency

A YZXStudio USB 3.0 Power Monitor was used to measure power consumed by the Pre-Processing and Sensing tier together while experimenting, as in a

more advanced prototype, they would be envisioned to be run on battery power. This was done by connecting the Arduino's USB cable to the Monitor, and the Monitor to a computer. It was calculated that the average power consumption was 51 mA at 4.92 volts with a sample rate of 1 Hz, while continuously outputting data. This power consumption did not vary significantly between sample rates, with the consumption increasing < 0.8 mA with the sample rate being set to 8 Hz.

To determine the draw from the Passive Infrared Sensor (PIR) and Thermal Detector Array (TDA), we disconnected all sensors from the Arduino, and ran the power measurement again. The same code was run on the Arduino. This time we received a result of 45 mA for 1 Hz, and 46 mA for 8 Hz. We can then conclude that the sensors themselves draw around 6 mA of power.

Bibliography

[1] Beltran, A., Erickson, V. L., and Cerpa, A. E. ThermoSense: Occupancy thermal based sensing for HVAC control. In *Proceedings of the 5th ACM Workshop on Embedded Systems For Energy-Efficient Buildings* (2013), ACM, pp. 1–8.

[2] University of Waikato. Weka. `http://www.cs.waikato.ac.nz/ml/weka/`. Retrieved March 10, 2015.