CHAPTER 1

Evaluation

Now that the sensing system prototype has been realized, we are now able to perform experiments involving the system. In this chapter we create a set of experiments to test the sensor's properties and come to conclusions as to their effect on our ability to detect occupants. We then outline a process from taking raw sensor data and performing occupancy predictions with it. Using that process, we then devise a set of experiments involving recording occupancy data, and finally we process said data and produce accuracy results.

1.1 Sensor Properties

In order to best utilize the Melexis MLX90620 (*Melexis*), we must first understand the properties it exhibits, and their potential affects on our ability to perform person related measurements. These properties can be broadly separated into three different categories; bias, noise and sensitivity. A broad range of data was collected with the sensor using various sources of heat and cold to determine these properties.

1.1.1 Bias

When receiving no infrared radiation, the sensor should indicate a near-zero temperature. If in such conditions it does not, that indicates that the sensor has some level of bias in its measurement values. We attempted to investigate this bias by performing thermal captures of the night sky. While this does not completely remove the infrared radiation, it does remove a significant proportion of it.

In Figure 1.1 and Figure 1.2 the thermal sensor was exposed to the night sky at a capture rate of 1Hz for 4 minutes, with the sensing results combined to create a set of means and standard deviations to indicate the pixels at "rest". The average
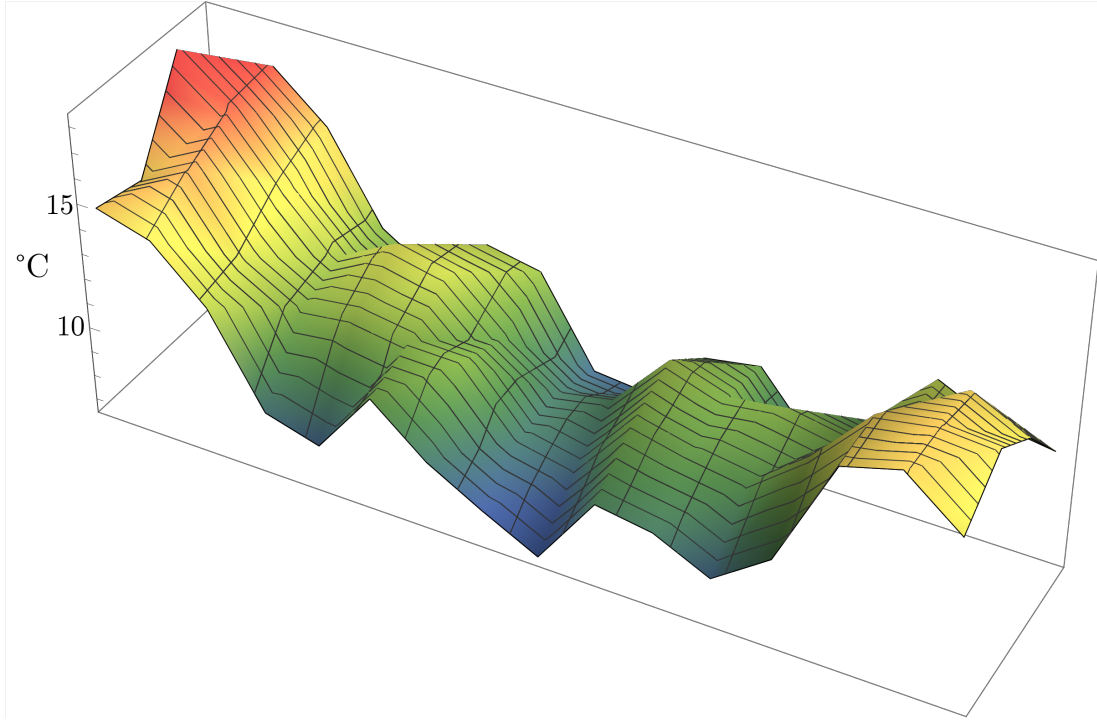
Figure 1.1: Plot of pixel means at rest

temperature detected was 11.78°C, with the standard deviation remaining less than 0.51°C over the entire exposure period. The resultant thermal map shows that pixels centered around the four "primary" pixels in the center maintain a similar temperature around 9°C, with temperatures beginning to deviate as they became further from the center.

The most likely cause of this 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 these sides are at an ambient temperature higher than the measurement data (as they 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. Such issues with temperature could be controlled for using a device that cools 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 does not dramatically change throughout capture.
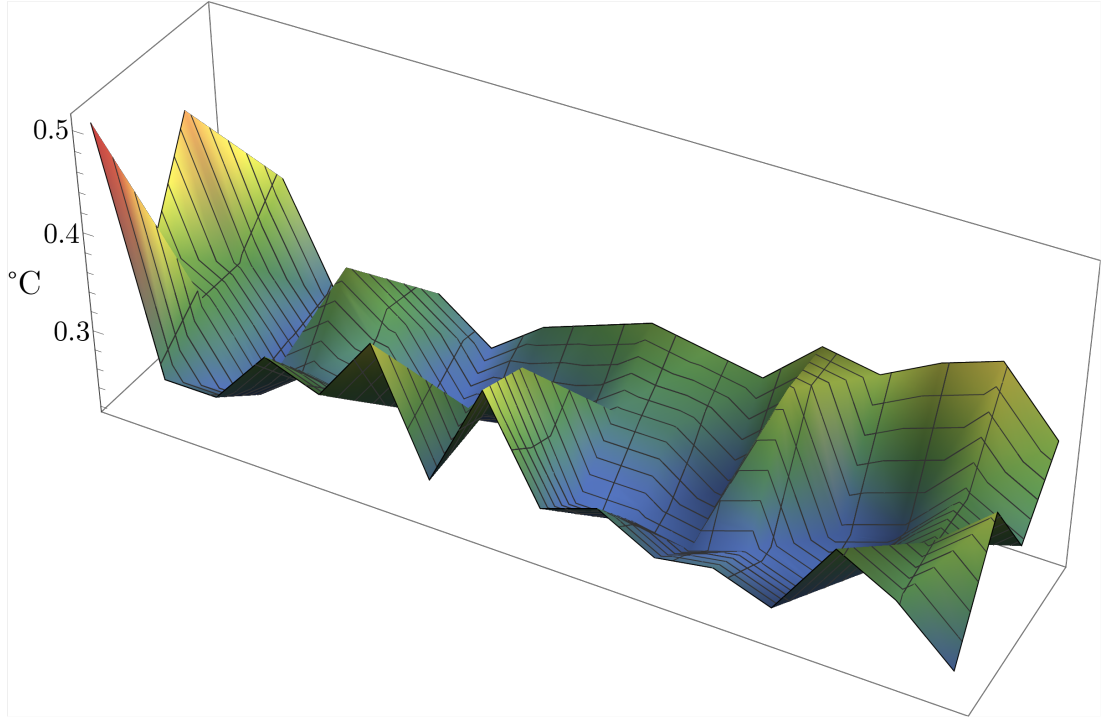
Figure 1.2: Plot of pixel standard deviations at rest

1.1.2 Noise

One of the features of the *Melexis* is the ability to sample the thermal data and a variety of sample rates between 0.5Hz and 512Hz. However, it was noted in early experimentation that a higher sample rate resulted in an increase in the noise contained within the resultant images. As our experiments focus on separating objects of interest from a thermal background, it is important to determine the maximum level of noise tolerable before our algorithms are unable to separate the background from the objects of interest.

Figure 1.3 plots one of the central pixels of the sensor in a scenario where it is merely viewing a background (shown in green), and when it is viewing a person (shown in red), at the 5 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 determine 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, it is likely that the ability for the sensing system to disambiguate between objects of interest and the background would diminish.
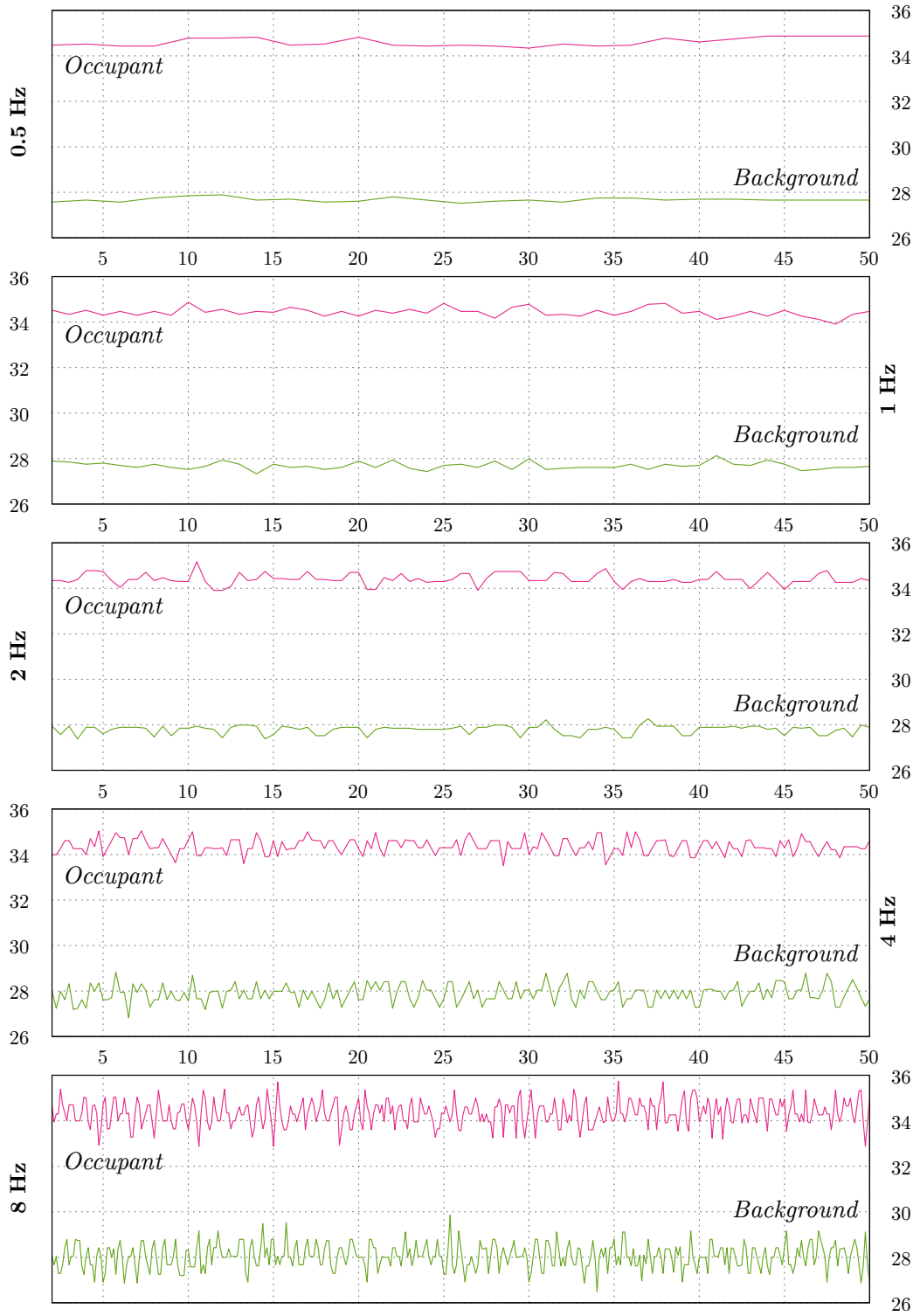
3

Figure 1.3: Noise levels of occupants and background at various sampling speeds

However, in the current project, even the slowest sampling rate of 0.5Hz is sufficient, as occupancy estimations at a sub-second level present little additional value and would require significant reforms in the efficiency of the software used. Therefore we conclude that sensor noise is not an issue at our current sampling rates.

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, Figure 1.4 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 1.5 shows a smoothed graph of the temperatures of the sensor top row's six centre pixels as a hot object is moved from left to right at an approximately similar speed. One of the most interesting phenomena in this graph is the apparent extreme variability of the detected temperature of the object as it moves "between" two different pixels; there is a noticeable drop in the objects detected temperature. Further analysis of each of the pixel's lines on the graph shows each pixel exhibiting a bell-curve like structure, 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 less favorable 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 elements field of view, the same happens in reverse.

While interesting, this phenomenon has little consequence to the effectiveness of the techniques used, as in experimental conditions the sensor will not be sufficiently distant that humans could be detected as single pixels. However, this phenomenon could be leveraged in future work to perform sub-pixel localization, discussed later on.
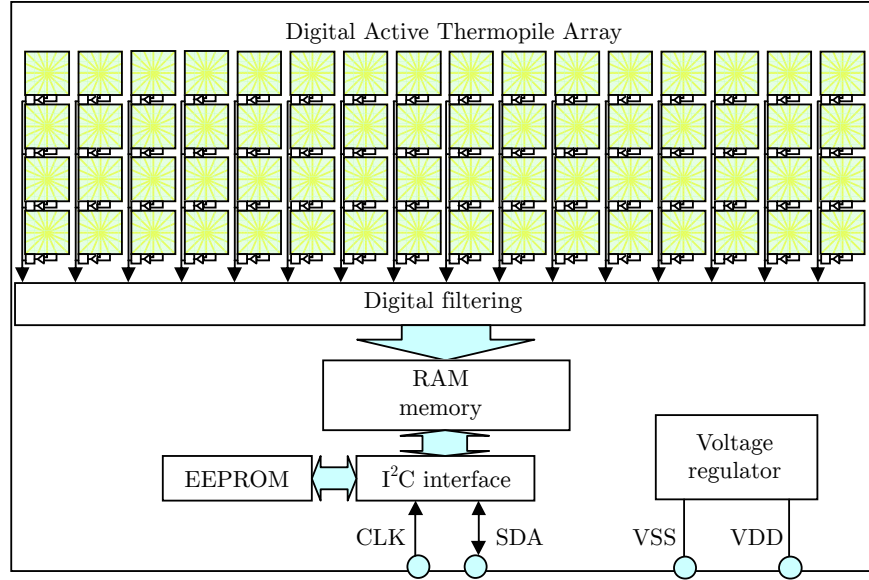
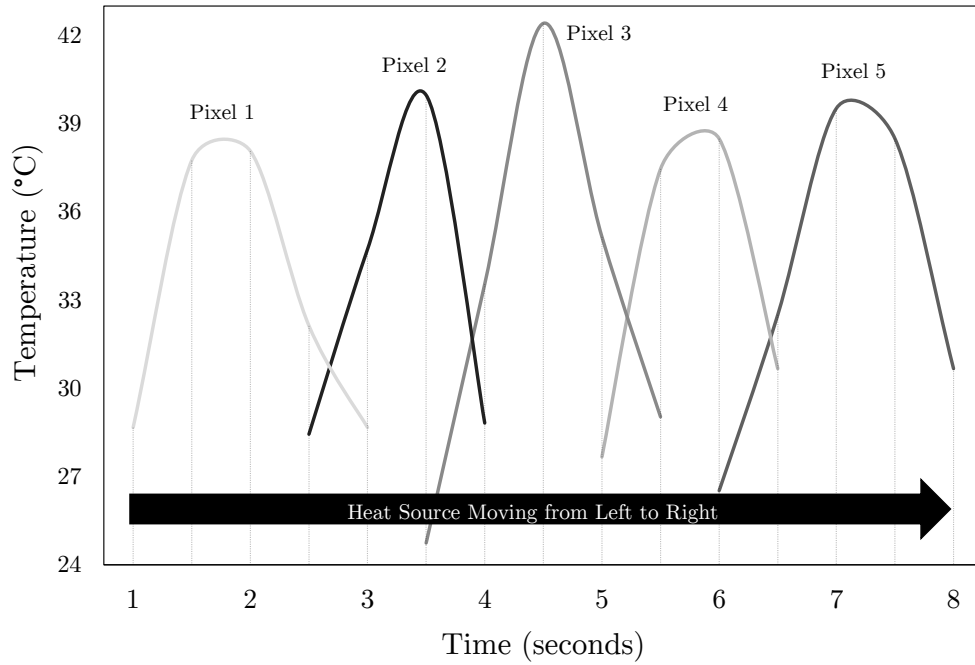Figure 1.4: Block diagram for the *Melexis* taken from datasheet [2]

Figure 1.5: Measure of top-centre 6 *Melexis* pixels as hot object moves across
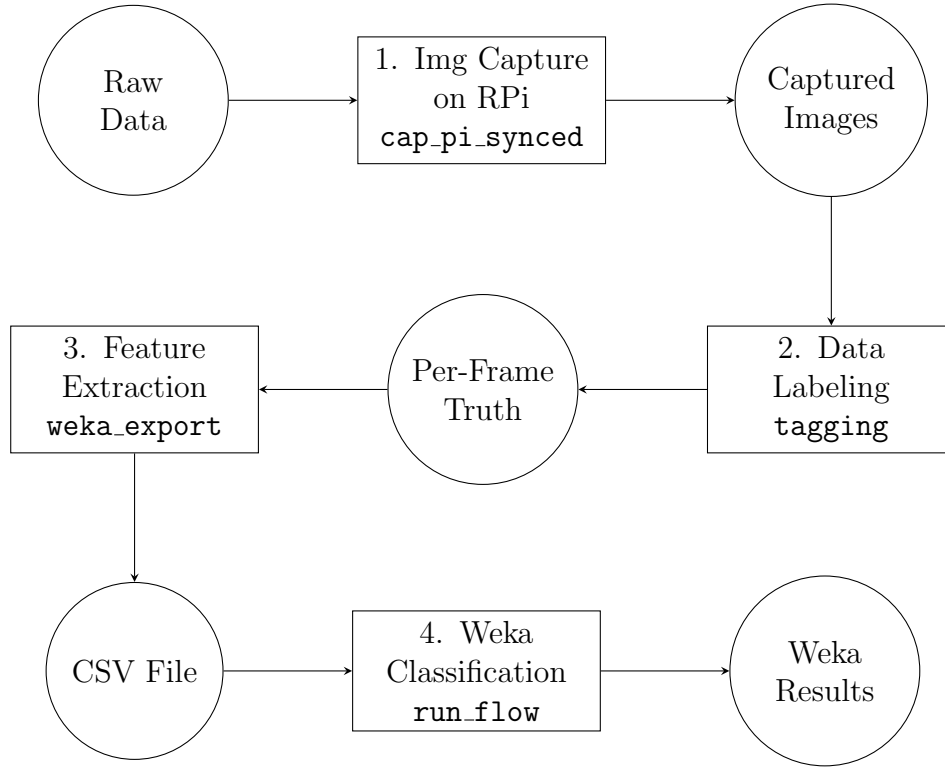
Figure 1.6: Visualization of processing steps

1.2 Classification Accuracy

With the prototype now fully operational, 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 human counting 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 experiments were devised, each of which had its data gathered and processed in accordance with the same general process, outlined in Figure 1.6 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. By default, it always captures at 2Hz. 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 the sending of 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 inside a file named `output_thermal.hcap` containing the thermal capture, 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 more powerful computer 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 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, making the process more efficient.

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 [3], which provides easy access to a variety of machine learning algorithms and the tools necessary to analyze their effectiveness.

To enable the use of Weka, we export the ground truth and extracted features to a Comma Seperated Value (CSV) file for processing. `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 algorithms, but we choose a specific subset of algorithms based on those present in the Thermosense paper [1], as well others that we believe adequately represent the different approaches to classification.

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"`

Decision Tree	Nominal	`weka.classifiers.trees.J48` `-C 0.25 -M 2`
Entropy Distance	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 classifications

To help maximize the efficiency of the classification task, we use the Weka Knowledge Flow constructor to generate an encompassing flow that accepts an input CSV file of the raw data, and performs all resampling, numeric and nominal classification, returning a text file with the results of each of the different

classification techniques run. The knowledge flow's struture can be seen in . Additionally, the input and output elements of this flow are set to the environmental variables `UnifiedFlow.InputCSV` and `UnifiedFlow.OutputCSV`, a Jython script, `run_flow.py`, then sets those environmental variables to input and output file names, then calls the flow using Weka's Java API. After this is complete, the script then runs a series of regexes on the output text data to generate summary spreadsheets with the relevant 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 deleted in `npeople`'s columns. For those tests that are "numeric," `npeople` is left unchanged, as by default all CSV import attributes are interpreted as such. For all tests where not specifically instructed, 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 cause the classification results to be affected. 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 Passive Infrared Sensor (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 1.7 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 script

1. (Remained standing) One person walks in, stands in center, walks out of frame. (sub-experiment 1)

2. (Remained standing) One person walks in, joined by another person, both stand there, one leaves, then another leaves. (sub-experiment 2)

3. (Remained standing) One person walks in, joined by one, joined by another, all stand there, one leaves, then another, then another. (sub-experiment 3)

4. (Remained standing) Two people walk in simultaneously, both stand there, both leave simultaneously. (sub-experiment 4)

5. (Sitting) One person walks in, sits in center, moves to right, walks out of frame. (sub-experiment 5)

6. (Sitting) One person walks in, joined by another person, both sit there, they stand and switch chairs, one leaves, then another leaves. (sub-experiment 6)

7. (Sitting) 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-experiment 7, 8)

8. (Sitting) Two people walk in, both sit there, both leave. (sub-experiment 9)

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 1Hz 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 occupancy using the script mentioned previously.

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 run the dataset against a set of classification algorithms we consider to be reasonable well known within the industry to get a general sense of how different approaches fare on our data. We have discussed each of these algorithms previously in the Literature Review.

a) View from side

b) View from above

Roof

150°

16.4°

2.6m

Occupant

Ground   0.7m

17cm

17cm
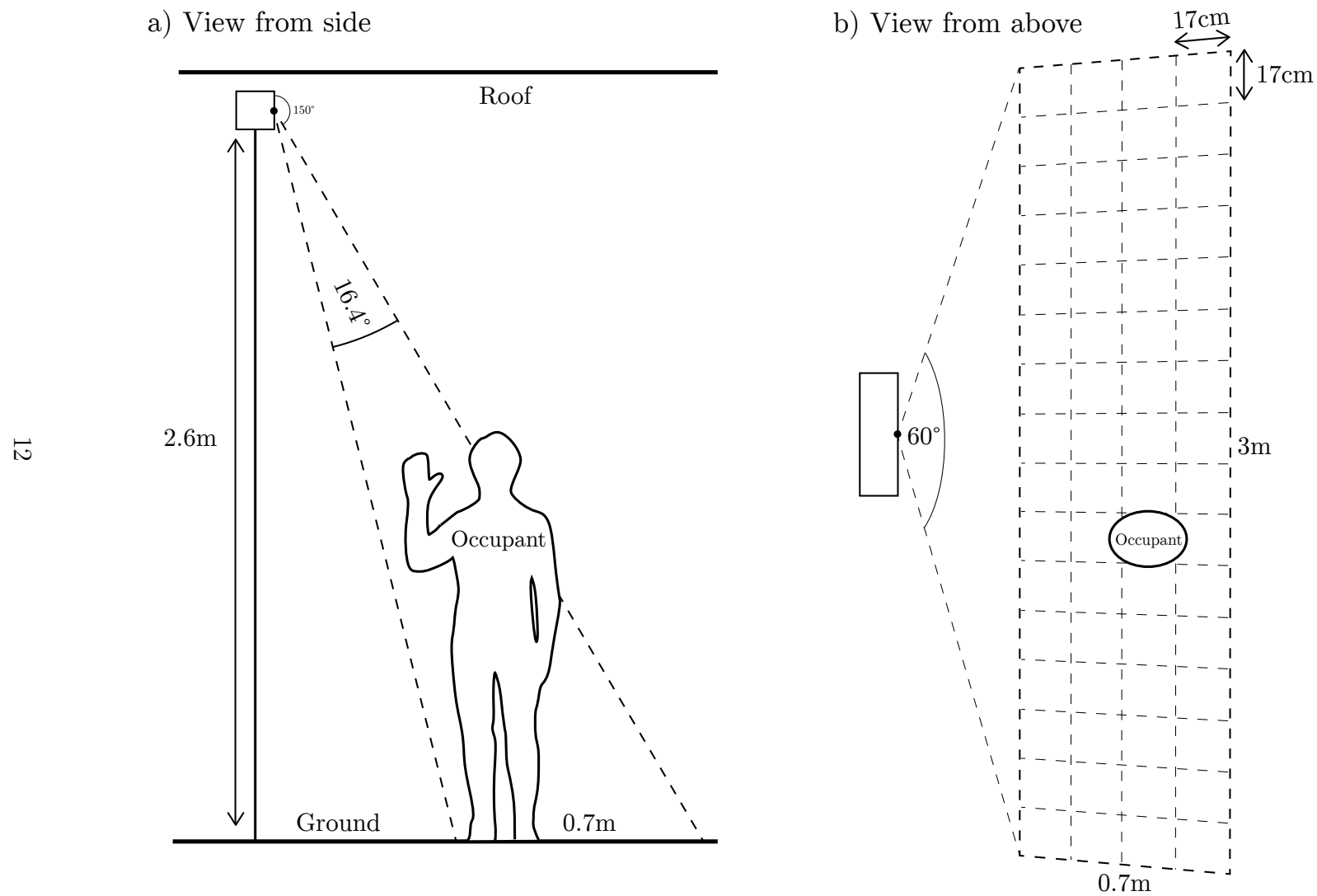
60°

3m

Occupant

0.7m

Figure 1.7: 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 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 what 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; they did not provide an R^2 value for that test, however they also did not provide any other % accuracy information either.

Because of this, four tests were performed overall to replicate the Thermosense results; two 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 performed consistently poorly when compared with Thermosense. 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), 0.592 (MLP) and 0.525 (Lin Reg). Additionally, our numeric KNN test performed worse than the 0-R benchmark for nominal tests, 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 better, and in some cases exceeded the RMSEs reported by Thermosense. Within our dataset, the K* and C4.5 algorithms were at the top, with accuracies of 82.56% and 82.39% respectively, and they both achieved RMSEs lower than the best achieved by Thermosense, with their 0.304 and 0.314 being a significant improvement on Thermosense's KNN RMSE of 0.346.

Following down the ranking, next our nominal MLP performed best, with an sufficient 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)

Classifier	RMSE	%	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	
Bayes	0.405	63.59	
0-R	0.442	49.74	

[1]: Included zero in training data

%: Percentage accuracy, for a nominal test

r: Correlation coefficient, for a numeric test

Table 1.2: Classifier Experiment Set Results

implementation achieved a relatively poor accuracy of 67.18% with an RMSE of 0.398, and finally the Naive Bayes (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 of the Thermosense numerical techniques, 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 the tiers on batter 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 1Hz, while continuously outputting data. This power consumption sdid not vary significantly between sample rates, with the consumption increasing < 0.8 mA with the sample rate being set to 8Hz.

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 1Hz, and 46 mA for 8Hz. 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] MELEXIS. Datasheet IR thermometer 16X4 sensor array MLX90620. `http://www.melexis.com/Asset/Datasheet-IR-thermometer-16X4-sensor-array-MLX90620-DownloadLink-6099.aspx`, 2012. Accessed: 2015-01-07.

[3] UNIVERSITY OF WAIKATO. Weka. `http://www.cs.waikato.ac.nz/ml/weka/`. Accessed: 2015-03-10.