Kiwi Observation System

Submitted to

Wentworth Institute of Technology

Department of Electrical Engineering and Technology

550 Huntington Ave
Boston, MA 02115
617-989-4590

Presented by

Sze “Ron” Chau
Benjamin Li
Sam Mei
Alex Young

April 2013

Abstract

This report outlines a device designed to be used by a research laboratory in the study of little spotted kiwi populations. The client requires a device with the capabilities of weight measurement and image capturing with an emphasis on minimal misidentification. The Kiwi Observation System (KOS) accomplishes these constraints as well as introduces further specifications that will be beneficial to accurate photographic data acquisition of the research subject. A focus on a field-ready, cost-effective design is accomplished through the use of open source hardware and software. The choice of open source components such as a Raspberry Pi computer and an Arduino microcontroller maximizes flexibility in the design by allowing the use of different camera and weight measurement modules. Currently the KOS remains in a proof-of-concept stage due to programming and hardware integration challenges. The ultimate goal for the KOS is successfully bringing the device into a functional state, and delivering it to researchers for data acquisition of little spotted kiwi populations.

Table of Contents

1. Introduction

1.1 Background

With an estimated population of 2000 individuals, the little spotted kiwi (apteryx owenii) is the smallest and second rarest of the flightless kiwi species native to New Zealand [1]. The range of little spotted kiwi populations markedly decreased with the arrival of human settlers within their habitat regions. Recent conservation measures have reversed the population decline of the little spotted kiwi. The International Union for Conservation of Nature and Natural Resources has the species currently classified as Near Threatened as of 2008 [2]. Conservation actions proposed by the IUCN calls for regular monitoring of all little spotted kiwi [3].

The little spotted kiwi is a predominantly nocturnal species [4] that populates a range restricted to seven offshore islands featuring minimal predators. The little spotted kiwi is also noted to maintain a preference towards wet environments. As part of the ongoing research of the little spotted kiwi, a research lab expressed interest in obtaining imagery data of this species that meets specific requirements.

1.2 Problem Definition

As of 2013, methods for acoustic identification [5] exist for research of little spotted kiwi populations, however no suitable device has been designed capable of photography-based monitoring. A research laboratory studying the population of the little spotted kiwi has identified this deficit, and requires such a device for the purposes of analyzing the health of the subject populations. The constraints of this device, as determined by the client, include the ability to determine if a bird weighs in excess of 1.75lbs and to capture a photograph if the bird meets the weight criteria. A secondary requirement requested asks for the design of a method to distinguish a little spotted kiwi from other species, and to capture a photograph if the bird meets these identification criteria.

The design must take these traits into consideration, or else the device would be severely handicapped for fieldwork. Maintaining portability is an objective that will allow the device to be brought into field study areas with little to no disturbance to the sites. This added priority will serve to create a field-ready prototype and limiting weather-related concerns.

2. Literature and Research

2.1 Overview

Literary research for this design split into two distinct avenues: Computer vision and weight measurement. These avenues allows for our design to identify and photograph an object; and identify an object's weight in pounds (lbs.).

2.2 Computer Vision

The development of an image processing technique that fits our objectives involves object recognition within the field of computer vision. Computer vision is the science of imagery acquisition and interpretation through the use of computers [6]. Object recognition describes the task of locating, or recognizing, specific objects within images. In the case of our design, object recognition tasks are incorporated to evaluate images arriving from a camera unit for stills containing the little spotted kiwi. The object recognition methods that were considered includes Haar-like features, blob detection, and template-based matching.

2.2.1 Haar-like Features

Haar-like features gets their namesake due to similarities to Haar wavelet transforms. It is an object recognition algorithm that seeks out common patterns by calculating spatial relationships of information (such as contrast) of the target object [7]. These patterns that the Haar-like algorithm checks are rectangular features that specific objects all have, such as the distance between a human's eyes, or nose (Fig. 1).

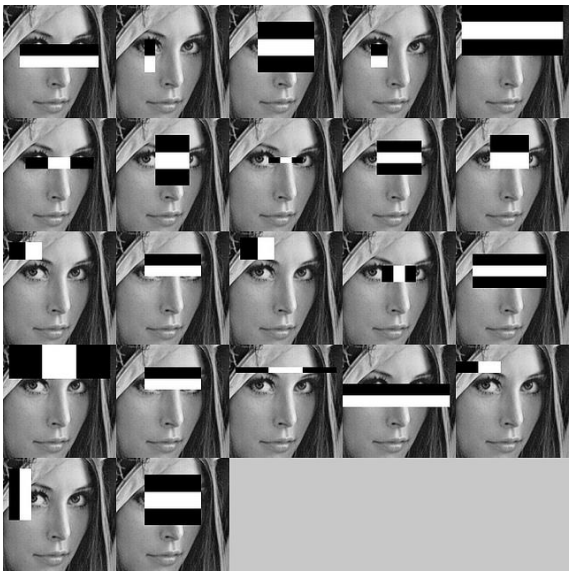


Figure 1: Haar-like cascade algorithm represented with rectangles.

While Haar-like feature detection is fast and retains a high degree of accuracy for animate and inanimate objects [8], a template database of positive and negative image samples is required

to train a classifier to a target object. This stipulation forced Team 6 to discard using Haar-like features due to the limiting storage on the Pi for a classifier database.

2.2.2 Blob Detection

Blobs refer to regions of similar pixels in an image. Blob algorithms locates these regions and assigns a category to it based on light or color values [9]. This method requires segmentation of the target image, which involves in the removal of everything outside the region of interest. Depending on the target environment, blob analysis is subject to noise in the form of mis-categorized blobs. The deployment of a blob analysis algorithm would require significant preprocessing of our target object. The near-nocturnal nature of little spotted kiwis results in captured frames of low contrast. While blob analysis would still complete the set objectives, the added preprocessing stage is inefficiently handled by the Pi. This algorithm was bypassed since the lighting conditions will be constantly sub-optimal for our intended period for image acquisition.

2.2.3 Template-Based Keypoint Detection

Keypoints are defined as simple but prominent object/points, such as corners, in an image that remains stable through light-level and positional changes [10]. For example, a corner keypoint would be a section in an image that contains significant changes of intensity in at least 2 directions. Template images are used to cross-check the target image for keypoint matches. This method locates keypoints and detects matches in the target image based on a user-defined threshold. This algorithm is selected for implementation into the KOS due to its effectiveness under low-light conditions. Keypoint matching does require template images for comparison, so there is a requirement for images of the little spotted kiwi at different angles.

2.3 Weight Measurement

Team 6 looked into the construction of common household digital bathroom scales for the objective of obtaining a little spotted kiwi's weight. These bathroom scales contain load cells that are connected to a printed circuit board (PCB, Fig. 2). This section looks into the possibility to repurposing one of these bathroom scales by removing the wiring to the PCB and connecting it to an Arduino Uno (details on Arduino in section 3.5.1).

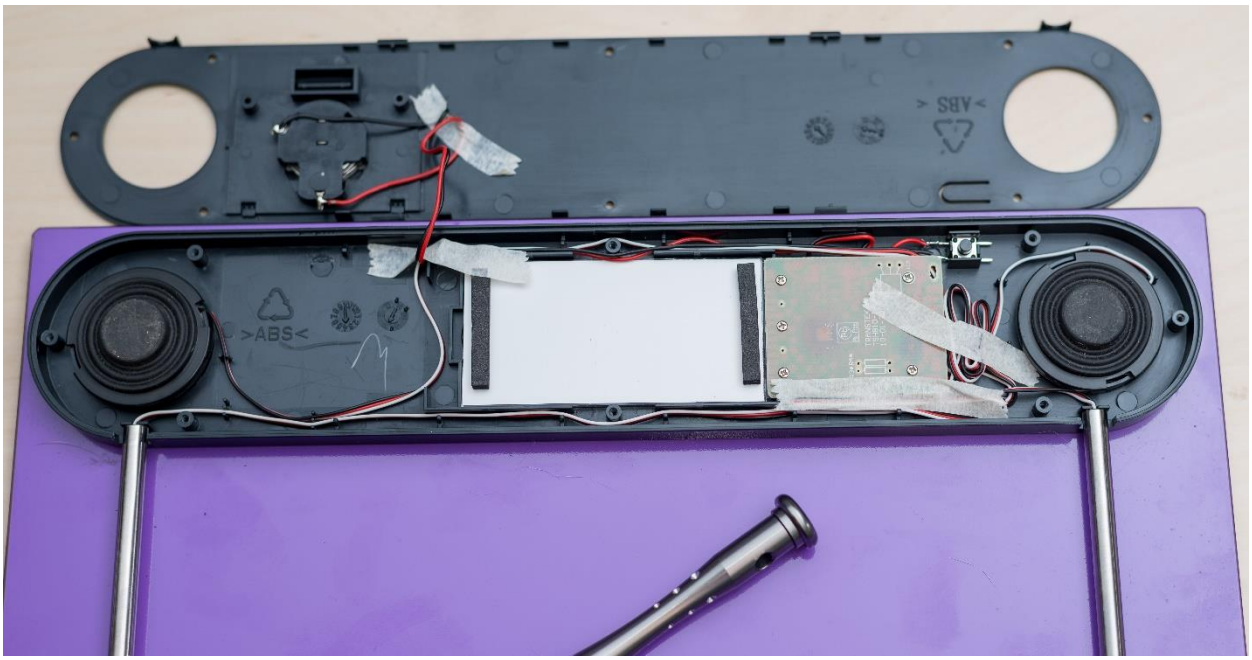


Figure 2: Repurposed digital bathroom scale. The enclosure contains load cells, battery, a push button, an LCD screen, and a PCB.

2.3.1 Load Cells

Load cells are devices that detect weight and converts the information into a signal. In the case of digital bathroom scales, the load cells contain strain gauges that changes resistance when weight (strain) is applied onto it [11]. When mounted onto a non-conductive load cell, force that is applied onto it results in a slight compression, which raises or lowers the resistance in proportion to the force applied. Utilizing a Wheatstone bridge circuit [12], this resistance change can be converted for a voltage change. Using the Arduino, we can intercept this signal, and convert it to a weight through using a linear formula developed through measuring known weights for a voltage value. Arduino Uno can take the analog input (weight on the scale's platform) and convert it to a digital value in the form of voltage. We have to be aware of the limitation of 10 bits of Arduino's Analog to Digital Conversion (ADC), meaning,

$$\frac{5\text{V}}{1024\ \text{steps}} = \frac{4.9\text{mV}}{\text{step}} \quad (1)$$

This limitation is must be considered as the scale will be driven using Arduino's 5V operating voltage [13]. This, however, results in a low sensitivity to applied changes. In order to heighten this sensitivity, we turn to an instrumentation amplifier. Figure 3 shows one of the four load cells used.

Figure 3: One of the load cells on the scale.

2.3.2 Instrumentation Amplifier

Including an instrumentation amplifier integrated circuit (IC) in our Scale structure will allow for a higher output that would otherwise remain in an mV range. This type of IC receives voltage signals as inputs and releases a linearly scaled, amplified output [14]. It is commonly used to raise the voltage signal from strain gauges. Using resistors to set the gain, the voltage signal coming from the load cell can be amplified before Arduino receives it.

3. Design Overview

3.1 Team 6 Solution

In consideration of the objectives set forth for the design, Team 6's solution focused on developing a highly portable, cost-efficient device for field use. The selection of hardware and software reflects focus on the portability aspect: Palm-sized computers and microcontrollers; and open source computer vision software. The Kiwi Observation System is a Team 6 design that incorporates these elements in order to fulfill the research lab's criteria.

3.2 KOS Overview

The KOS is separated into 2 main structures: Vision and Scale (Fig. 4). These structures are handled separately, with the Raspberry Pi acting as the project platform and Vision structure, and the Arduino Uno maintaining the Scale structure. A waterproof Pelican case contains and protects the Pi and Arduino units from the environment.

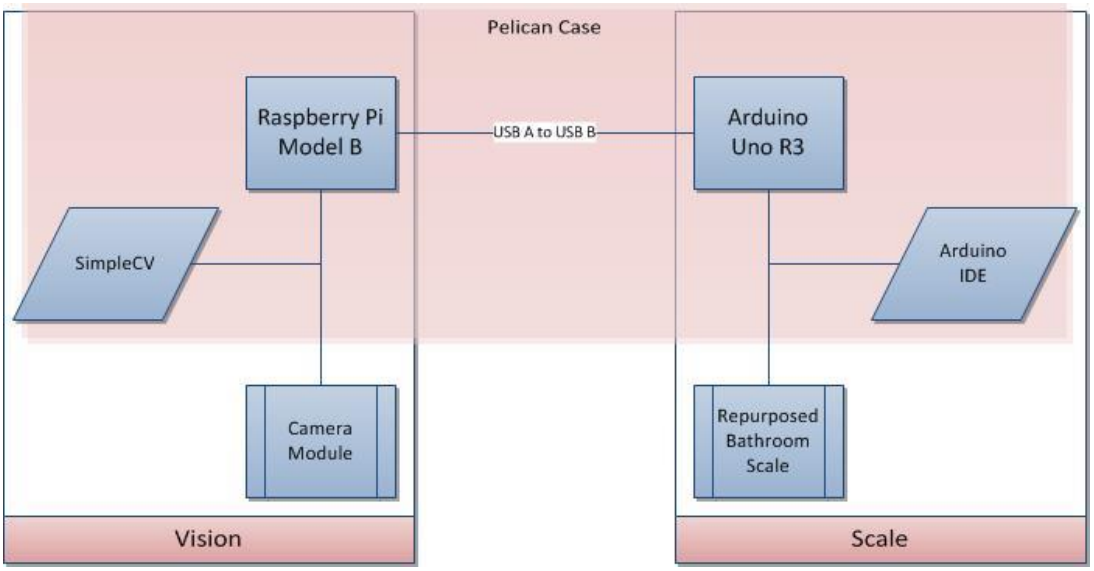


Figure 4: KOS Structures: Vision and Scale

Figure 5 shows the method in which the KOS accomplishes the objectives. From the moment an object steps on the scale, KOS Cam, our concept application, runs through the steps outlined in the flowchart.

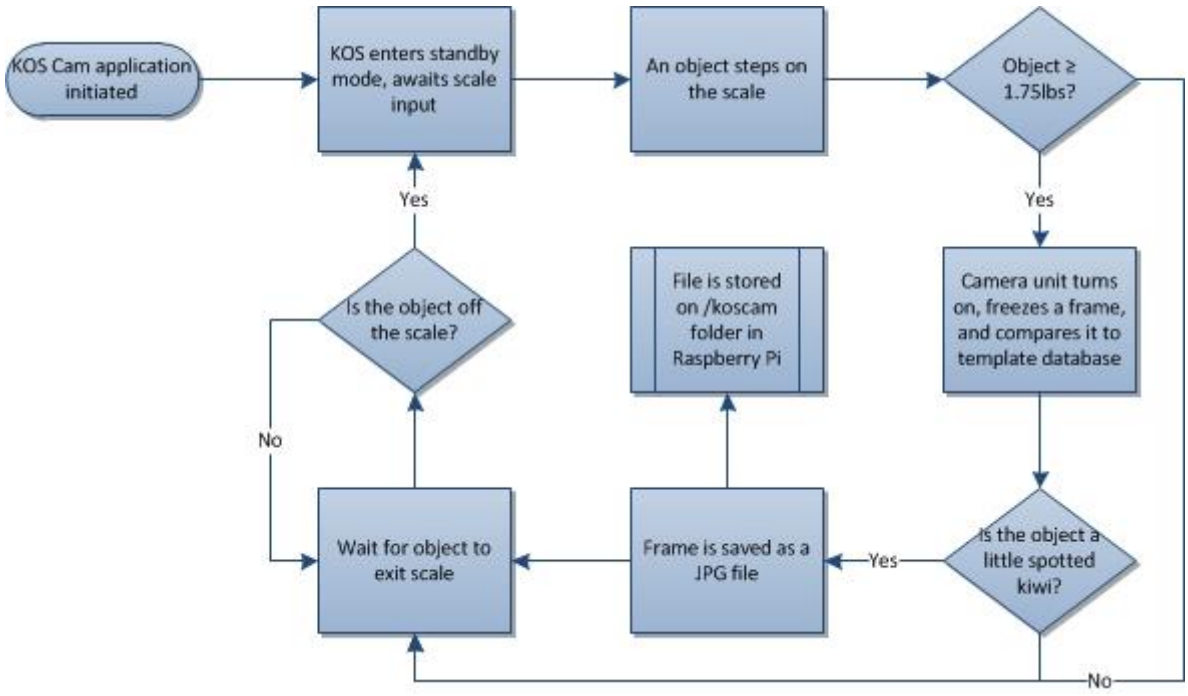


Figure 5: Flowchart showing the steps of KOS Cam.

3.3 Project Platform: Raspberry Pi

The Raspberry Pi Model B is a small, single-board computer (Fig. 6) that allows for a choice of Linux-based operating system distributions [15]. Along with obtaining the camera unit's input, the Raspberry Pi also acts as the project platform for the KOS. The KOS runs on the officially recommended OS: Raspbian, a Debian-based Linux distribution optimized for use on Raspberry Pi.

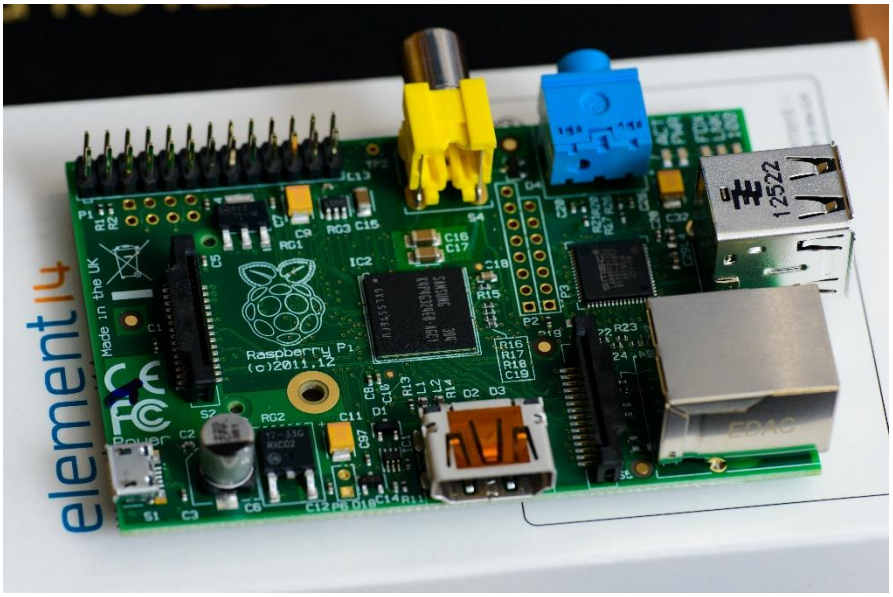


Figure 6: Raspberry Pi Model B

The selection of Raspberry Pi as the project platform factors in aspects of ease of use, cost, portability, and freedom for commercial use [16]. Raspberry Pi's official programming language is Python, and through LXTerminal, access to the command line is permitted. LXTerminal allows a user to work in Python by testing one line at a time. This is due to the Python programing language being designed as an interpreted language, meaning that compiling code isn't necessary prior to executing a script [17].

3.4 Vision

The Vision structure's hardware consists of the Pi and a camera unit (Fig. 7). The intended final design incorporates an Adafruit TTL serial camera module in a weatherproof housing, but due to the time constraint the current prototype uses a Logitech C615 webcam. Both camera units perform the same task of image acquisition, but the main benefit of using an USB-based webcam over a TTL serial camera is ease of use. To access the webcam on Pi, one simply attaches the USB cable on the camera to one of the Pi's USB ports.

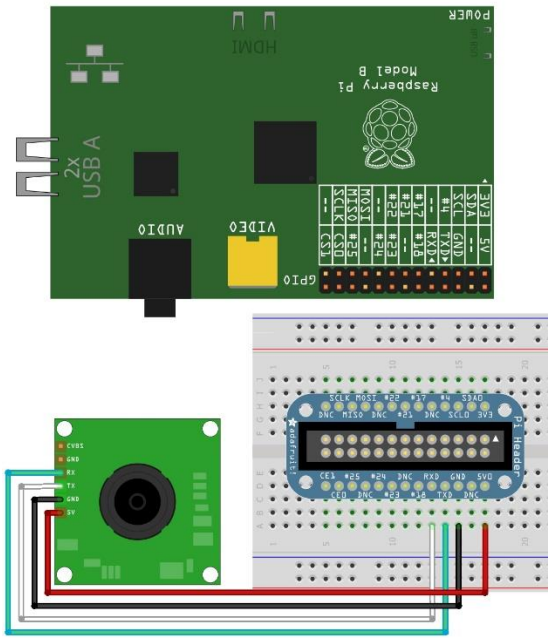


Figure 7: Fritzing Diagram of the Raspberry Pi and TTL Serial camera module. The module in this diagram lacks the weatherproof housing.

The benefits of using the TTL serial camera module over the USB webcam includes: Bypassing Pi's USB 2.0 bottleneck [18], included IR LED system for nighttime use, and a weatherproof enclosure. Since the TTL serial camera module uses the Pi's GPIO pins (specifically, the UART) to connect with each other [15], it would open up one of the USB ports. Though the TTL serial camera module hasn't been fully incorporated into the KOS prototype, the Logitech C615 webcam is capable in executing the Vision tasks, albeit to a slower extent. Both camera units are also comparable in size (Fig. 8) and price.

Figure 8: Logitech Webcam (left) and Adafruit TTL Serial camera (right) side by side comparison.

3.4.1 SimpleCV

The Vision structure's software utilizes SimpleCV framework. Simple Computer Vision, officially SimpleCV, is a collection of libraries and algorithms written in Python used for computer vision applications [19]. This open source framework allows the Pi to access an attached camera unit and apply object recognition tasks to the extracted images. The combination of SimpleCV and the Pi means that Python, the official language for both entities, will be the sole programming language for the Vision structure.

Of the multitudes of object recognition algorithms included with SimpleCV, Team 6 selected template-based keypoint matching as the method for the Vision structure (See 4.3.1 for details).

3.5 Scale

This separation was necessary due to the limitations of our hardware selection. Through KOS trials, the Raspberry Pi utilizes an average of 90% of its processing power in order to perform object recognition algorithms. Removing the duty of interpreting weight on the Pi allows it to focus on the data from the camera. Instead the weight scale is handled by an Arduino

Uno microcontroller that will only relay information to the Pi in the event that an object weighing greater than or equal to 1.75lbs is introduced onto the scale. The weight measurement unit is a repurposed Terraillon digital bathroom scale. The KOS utilizes this bathroom scale's platform and load cell configuration. The configuration is redirected to Arduino by way of breadboard.

3.5.1 Arduino

The Arduino Uno R3 (Fig. 9) is an open source prototyping platform that is based on an ATmega328 microcontroller to receive input from environmental sensors, interpret the input, and in turn outputs interactions according to the user's designs [13]. It has a programming platform based on the Wiring language, but it could also be controlled through the Pi.

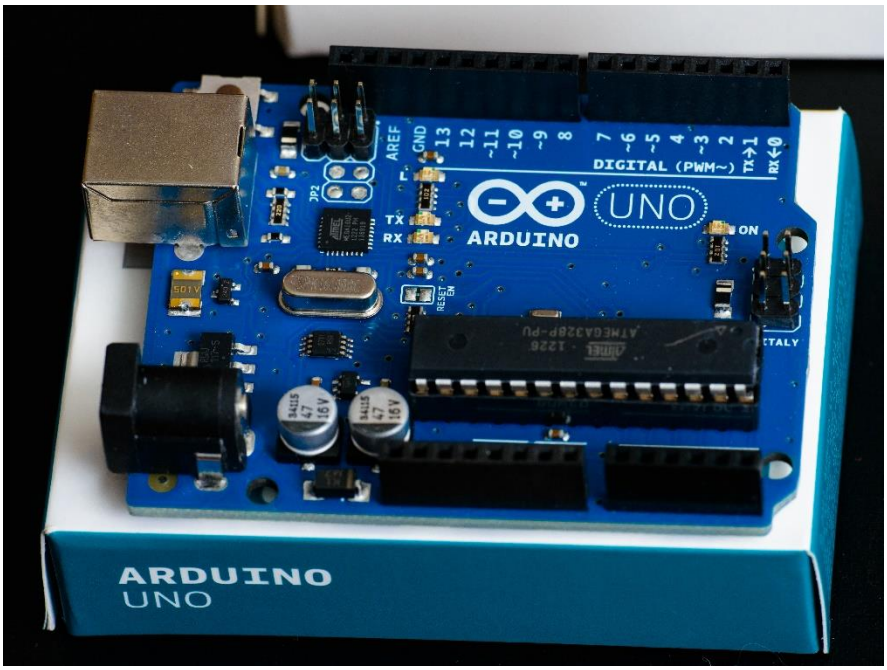


Figure 9: Arduino Uno R3

4. Evaluation

4.1 Overview

As part of the prototyping phase, different possibilities for the KOS were trialed in order to find the optimal configuration. Of the Vision structure, several algorithms were tested on the Raspberry Pi. For the Scale structure, an evolution from a Flexiforce sensor to a load cell configuration resulted in a lack of data for this section. As of now, the Scale structure is the primary portion that remains in a proof-of-concept state.

4.2 Mr. Samsa

In order to examine the KOS in a similar condition to the actual study environment, Team 6 enlisted the help of Mr. Samsa, a realistic hand crafted replica of a kiwi bird by Hansa Toys that closely match the little spotted kiwi in features (Fig. 10). Table 1 compares the measurements of an average apteryx owenii [20, 21] to our replica.

Figure 10: Image of a little spotted kiwi (left) compared to Mr. Samsa, a plush replica (right).

Table 1: Comparison of size between average little spotted kiwis and Mr. Samsa.

	Average Apteryx Owenii	Mr. Samsa
Height (in.)	15 in.	9.1 in.
Body Length (in.)	14-18 in.	10.6 in.
Bill Length (in.)	2.68 in. (males) 3.35 in. (females)	4 in.

While Figure 10 shows noticeable differences (color) between an actual little spotted kiwi to Mr. Samsa, the overall features (shape) remains the same. This allows Team 6 to test our KOS on a similar object. In addition to working primarily with outdoor images, the inclusion of Mr. Samsa simulates possible deployment conditions for our design.

4.3 Vision

Working with SimpleCV through the LXTerminal in Pi allows us to use Python's Interactive Interpreter. The benefit of this is the ability to trial single lines of code at a time through a command line shell. Figure 11 shows the process of creating an application that would save an image of the entire desktop. The upper left, pink-colored window shows the LXTerminal. All related code is located in Appendix D.

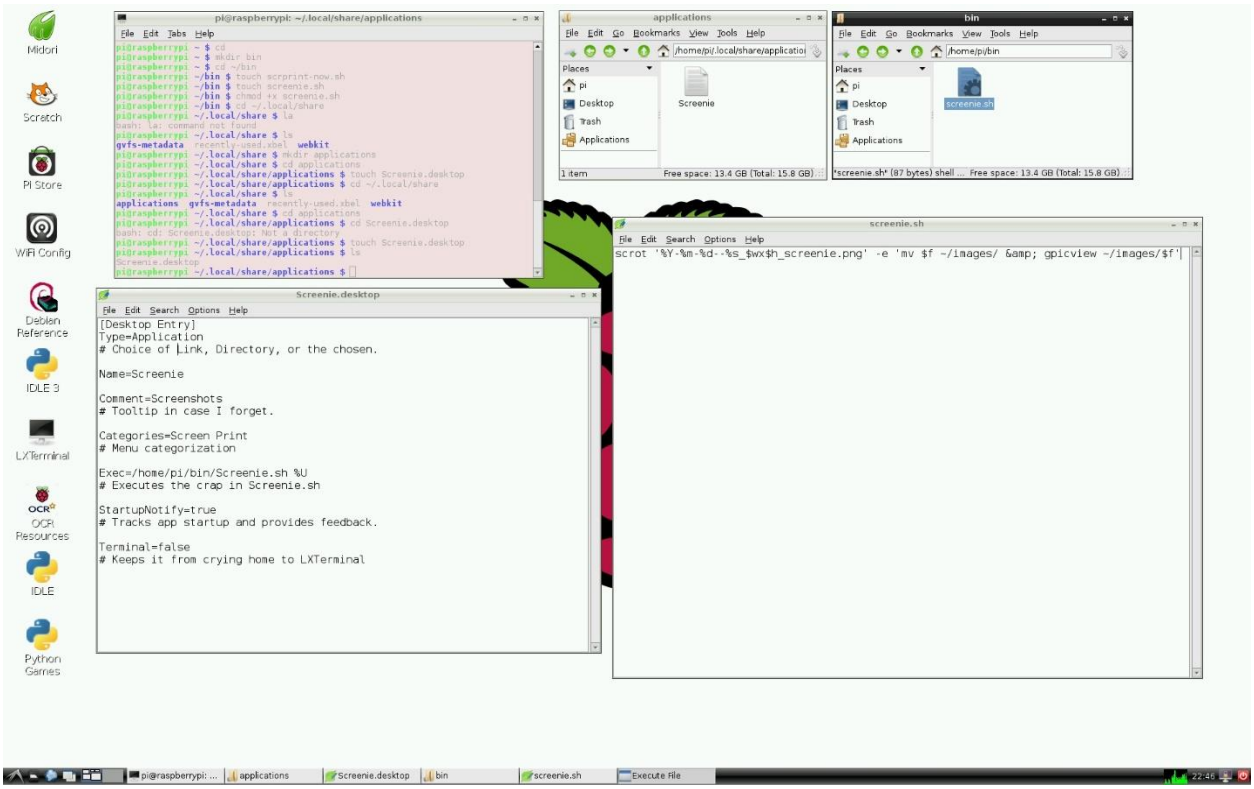


Figure 11: Creation of a screenshot application with the aid of LXTerminal.

4.3.1 Trialing Object Recognition Methods

The first method that was evaluated for use was Haar-like Features. SimpleCV supplies a cascade for human faces, so this was first used (Fig. 12) in order to determine the feasibility in moving forward with training a custom cascade. A simple program was written that printed messages depending on whether the algorithm located a face. Each time the algorithm located a face, the subject would cover or hide their face to allow the algorithm to seek for human faces. Uncovering the subjects face to the detection of it averaged around 1 to 2 seconds delay. Running this algorithm on a 700MHz processor, with a real-time 640x480 resolution USB webcam feed and averaging such a low delay is impressive. Presenting the system with various objects like a Pug, a Shiba Inu, and Mr. Samsa resulted in zero false positives. While the speed of detection was fast in real-time trials, limitations in the Pi's ability to custom train a cascade for the little spotted kiwi meant that other methods will be better served.

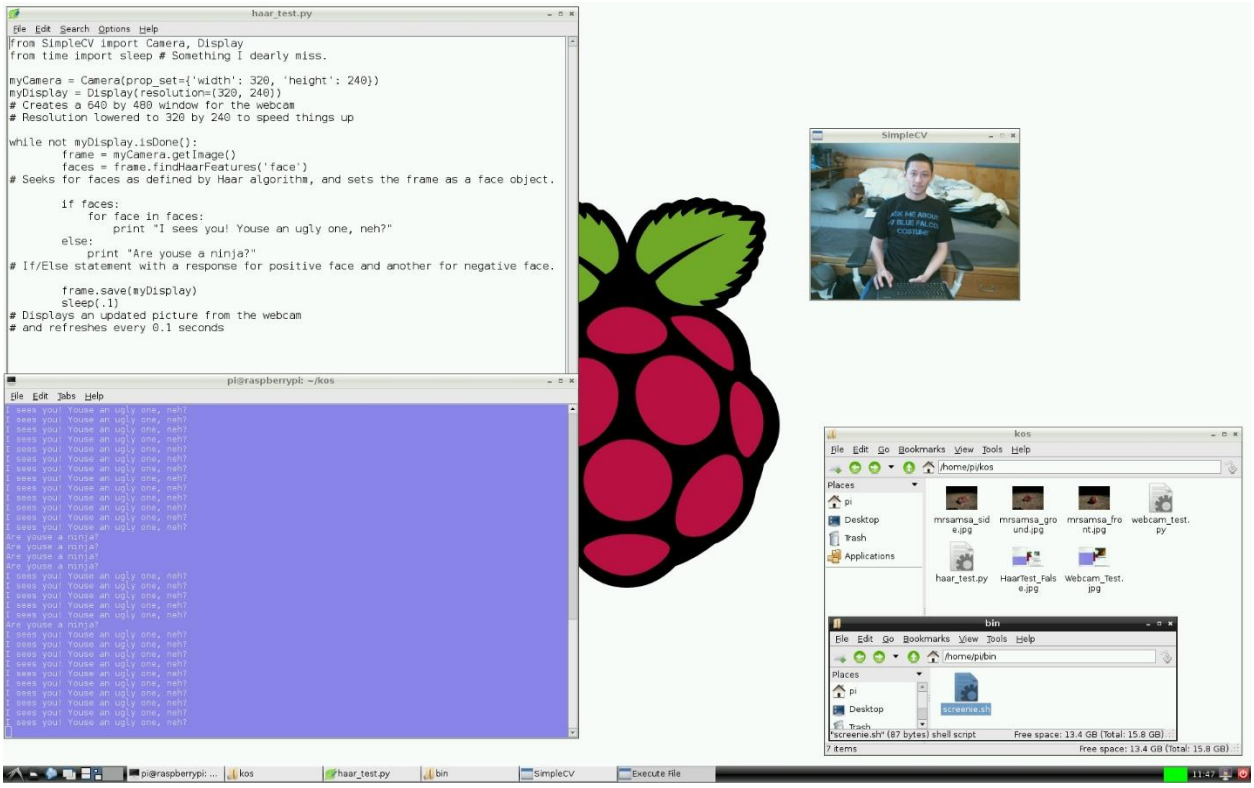


Figure 12: Evaluation of Haar-like features using a human face cascade.

Moving onto Blob Detection, the trials were based on outdoor images of Mr. Samsa. The reasoning is that blob detection would react differently with various backgrounds. Since the KOS is slated to be deployed in grassy, forested habitats favored by the little spotted kiwi, a grassy background was utilized to examine the effects of such heavy texture. Figure 13 shows two methods in which blob detection was applied: binderized (top SimpleCV window), and binderized and inverted (bottom SimpleCV window). Both methods resulted in a difficult time

for the algorithm in distinguishing between Mr. Samsa and grass. Hue-based methods are to be tested, but this has been postponed until the TTL Serial camera module has been integrated. Using the TTL Serial camera module will allow us to examine frames taken with the IR array activated. Further testing into blob detection without an IR image would result in data that might not apply to the KOS' intended use.

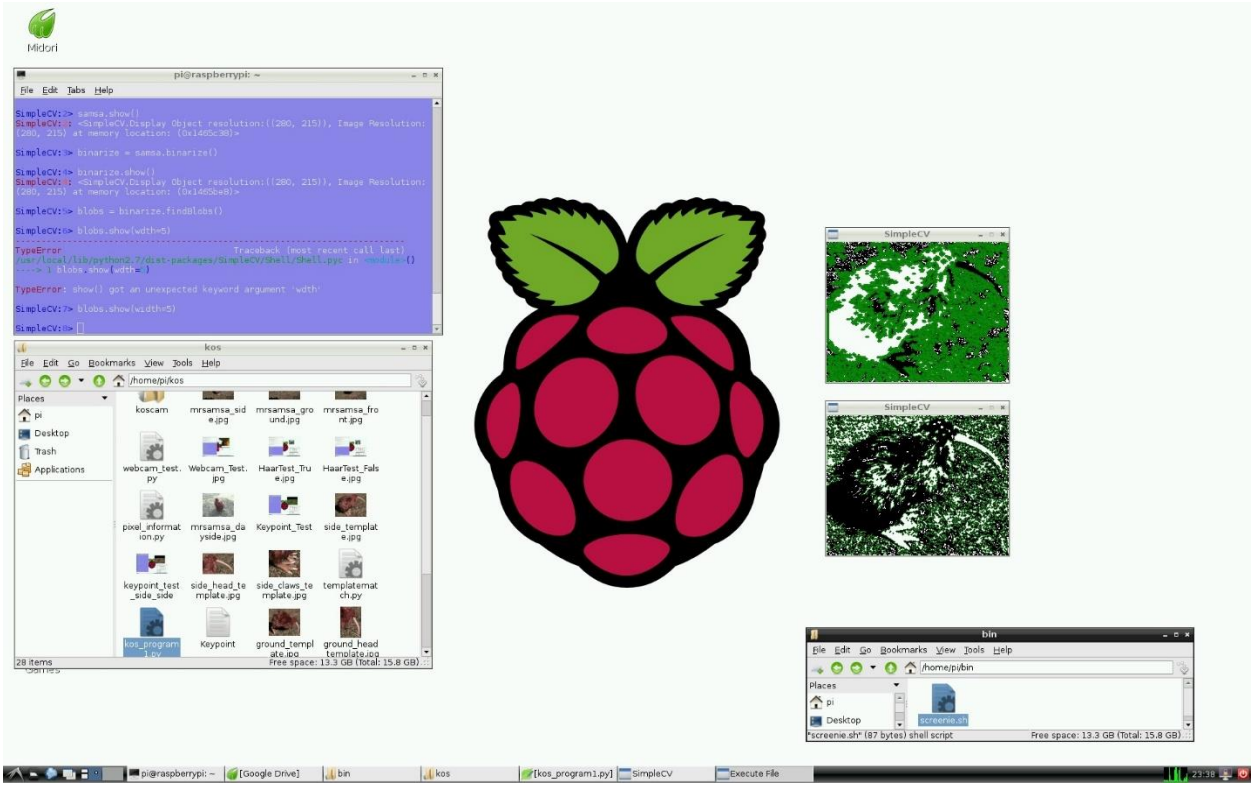


Figure 13: Blob detection methods. The top SimpleCV window shows a binarized image. The bottom one shows a binarized and inverted image.

Template Based Keypoint Matching was discovered to be the most optimal method for our purposes. While methods like blob detection required a stable lighting environment, keypoint matching worked in detecting Mr. Samsa in various light levels. For example, Figure 14 displays an image of Mr. Samsa that resulted in positive matches for a template image taken during daytime as well as nighttime. As long as the database is populated with images of the object at various angles, a detection will occur. In order to guarantee that the template database contains optimal images of little spotted kiwis, a method to incorporate images from the target location requires investigation.

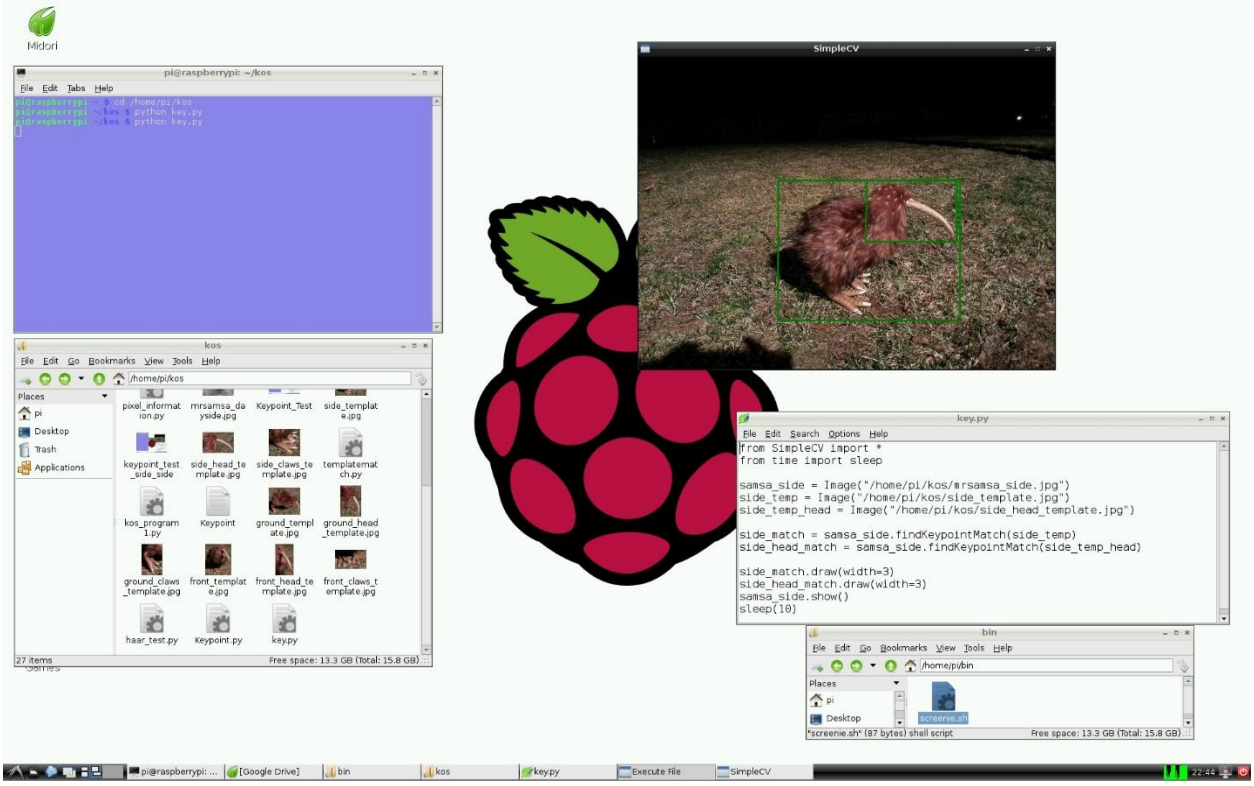


Figure 14: Template based keypoint matching using a still image of Mr. Samsa in a grassy environment.

4.3.2 Populating Template Database

The choice in a keypoint-based template matching method means that a database of templates will need to be constructed. While it would be simple to obtain random images online of little spotted kiwis, keypoint-based matching works best when the template images are as close to the target object as possible. Team 6 is in the process of incorporating an application (Fig. 15) within the PI that will capture still images when there is motion. These images will be saved in a folder for manual sorting. The images that contain a little spotted kiwi would be the optimal images to incorporate into the template database. The inclusion of this initial portion will allow the KOS to utilize the most accurate images to the deployment location as templates. This will serve to increase accuracy in keypoint matches.

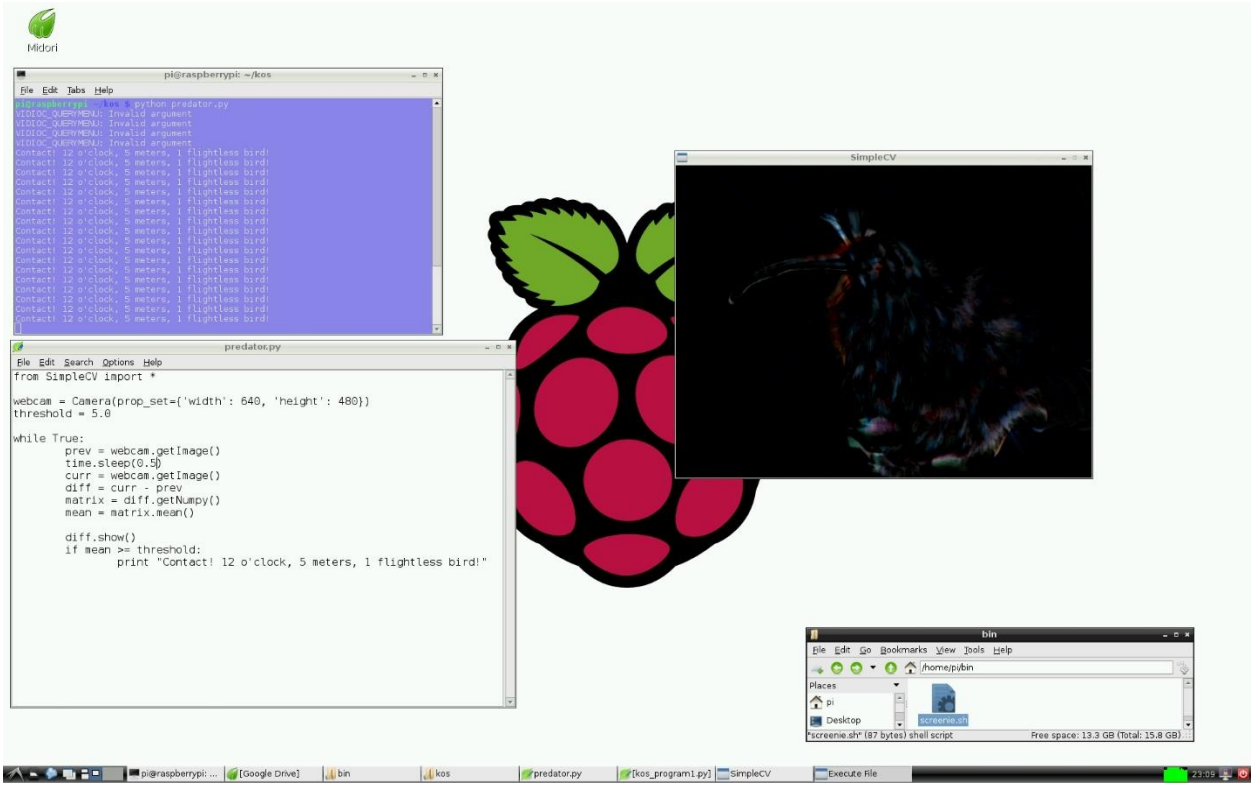


Figure 15: Motion sensing application.

4.4 Scale

As of present, the Scale structure is nonfunctional. This prevents any meaningful evaluation of this section of the KOS. An attempt to incorporate a different type of weighing system (Flexiforce sensor, Fig. 16) during the early prototyping stage consumed much of the time (1 week) devoted to the scale structure. When Team 6 agreed that the Flexiforce method wasn't practical (due to the need for platform design and construction), the Scale structure was reevaluated, and a choice was made to use a repurposed bathroom scale.

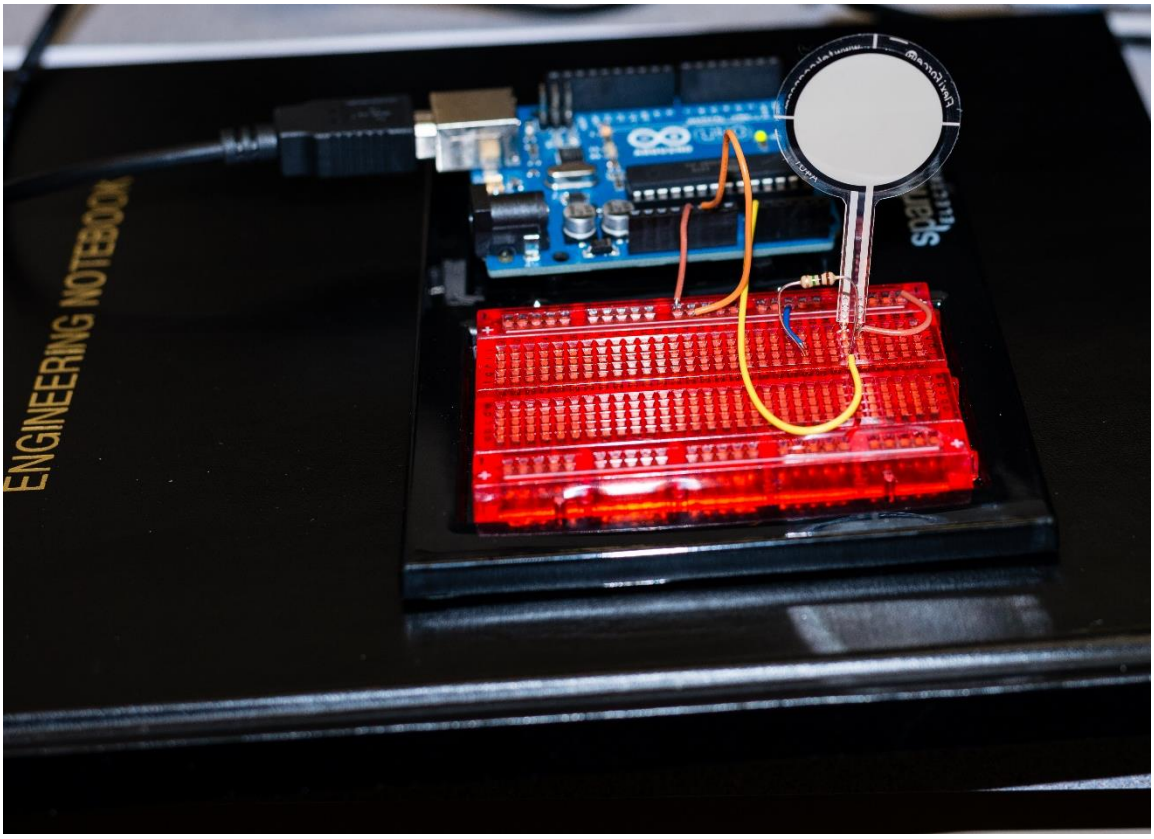


Figure 16: Early stages of the Scale structure included an evaluation of a Flexiforce sensor.

5. Discussion

5.1 Assessment

The KOS is currently an incomplete device that contains functional and non-functional elements. Figure 17 shows the most recent updates to the KOS. This is due to our group's strong emphasis on the portability and field use aspects. While the KOS remains at a proof-of-concept stage, Team 6 will continually put in work in order to raise it to a fully functional prototype. Through working with the separate structures that comprise the KOS, an analysis of this design has been completed, which includes: Strengths, drawbacks, risk and failure, competition, and future progress.

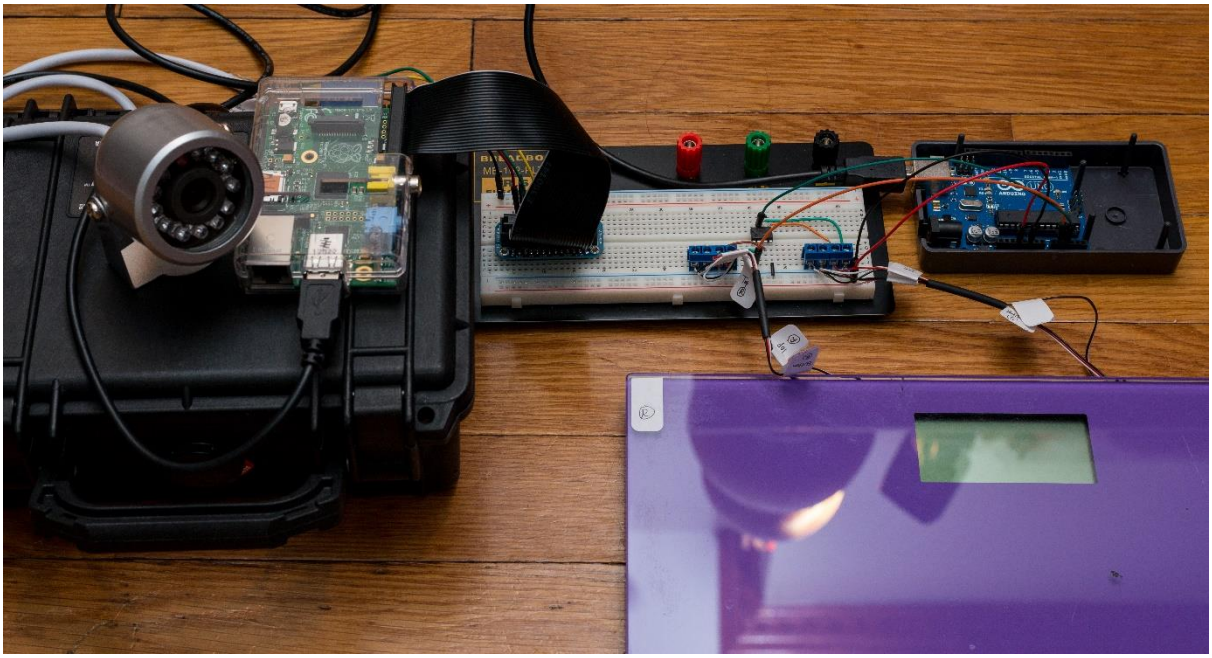


Figure 17: Current iteration of the Kiwi Observation System.

5.2 Strengths

The elements of this device was selected to allow for a cost-efficient design that would easily be carried in a rucksack by a researcher to the study site. Through traveling from Team 6 Headquarters to Wentworth Institute of Technology's Dobb's Lab, the entire KOS was transported with a 26 liters rucksack. Room remained within the ruck to carry various engineering notebooks, a textbook, and a laptop.

5.3 Drawbacks

The focus on a portable system means that power sources remain a weak point in the KOS. In order to power itself in the field, an additional purchase of a battery (USB battery packs designed to recharge cell phones would work) is necessary. This also leaves a question of how accessible the study location is to modern resources, that is, if the population of little spotted kiwis requires a day to reach, additional methods of powering the KOS will be needed. Methods

like solar (ineffective depending on tree canopy of the area) or a car battery (heavy, requires additional weather protection, and reduces portability) are possible choices for running the KOS.

A limiting factor to our use of a repurposed digital bathroom scale is the effect it would have on accuracy in field use. Strain gauges are susceptible to temperature changes, and since the KOS will be deployed outdoors, a concern for an accurate weight measurement will mean that a better Scale structure will have to be examined in future iterations [12].

5.4 Risk and Failure Analysis

Team 6 missed several milestones that were defined in our project proposal report. A functional prototype was expected to be the end result of this semester, but due to challenges in circuitry connections and programming syntax miscommunications.

5.4.1 Circuitry

The need for an amplifier for our Scale structure resulted in a non-functional outcome for our first objective. Each instrumentation amplifier that Team 6 attempted to incorporate into our design were met with difficulties in implementing into the KOS. This, however, isn't a challenge that's isolated to Team 6, as every other team discovered that their attempt to complete the first objective were also impeded by an amplifier. Figure 18 shows an attempt to integrate 2 INA125 instrumentation amplifiers into the Scale structure. A more detailed study of proper configuration will be necessary in order to bring the Scale structure to a functional state.

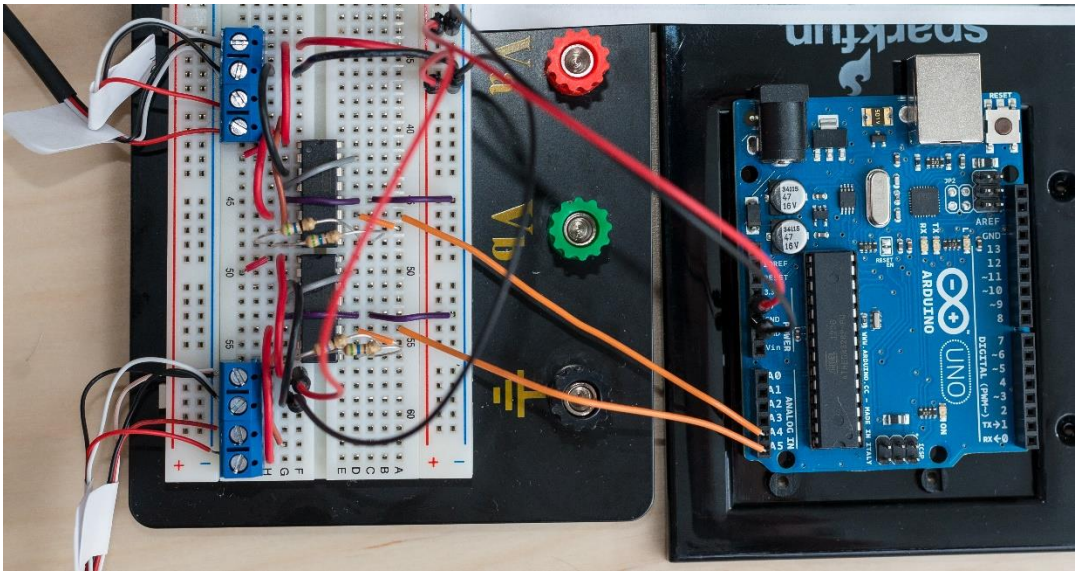


Figure 18: Circuitry for the Scale structure. The wires connected to the two terminal blocks leads to the repurposed digital bathroom scale.

5.4.2 Programming Syntax

The programming experience of each member of Team 6 is limited to this semester. The choice of a Pi and SimpleCV required all hands to invest time in understanding how Python works. This, along with the other priorities in the project and a lack of time, resulted in only a

basic understanding of Python. This can clearly be seen in that the KOS has no fully fleshed out KOS Cam (.py application file) for the researchers to utilize. While sections of code would execute properly in Python's Interactive Interpreter, syntax errors would occur when the tested sections were placed into one .py file. Of the challenges met within programming, notable ones include:

a. Program self-terminating when no matches are found for one template image. Instead of moving on to another template, the program concludes the keypoint matching process. This is an issue with our if/else statements being worded incorrectly.
b. Issues with populating a template tuple. Tuples in Python are immutable sequence data types. Currently the template images are written into our program line for line, making for a clustered and inefficient list that should be handled by a tuple.
c. Accessing the GPIO pins on the Pi. Accomplishing this would allow the KOS to switch over to the use of our TTL serial camera module, thus bypassing the USB bottleneck (and freeing a USB port for another device, such as a WiFi module for remote control.)

At this point the errors are being slowly rectified so that the KOS would perform as intended. Certainly an earlier focus on programming would have ironed out most of these challenges, but Team 6 erroneously focused on hardware selection as an early priority. In retrospect, a programming emphasis should have been pivotal in the early stages, as even if hardware changes occur, the code could have been adjusted for these changes.

5.5 Competition

A comparison of the KOS to our peer teams' interpretations show that there isn't a single complete prototype that achieves every objective set forth by the problem definition. Most teams focused on preparing a functional computer vision system at the cost of diverting time and energy away from the first objective: Identify the weight of an object and determine if it meets the design specifications. Some teams completely forgone the first objective in pursuit of an image processing method. Others have reversed their focus, entirely omitting a computer vision system from their design. While Team 6's Scale structure is currently non-functional, an even balance of time was given to each objective.

Out of all the designs featured at the Working Prototype Demonstration on 12 April 2013, no other design incorporated methods to introduce their devices into the environment that the little spotted kiwi inhabits. The KOS's design was analyzed at every stage for practical use in a wet, forested habitat. This emphasis on field usability certainly introduced drawbacks to the design, but maintaining the emphasis allowed the development of the only design that could realistically be deployed without major modifications.

5.6 Ethical Considerations

In accordance to the Code of Ethics published by ASME and IEEE [22, 23], every action taken by Team 6 is evaluated for any possible ethical concerns. While the project meets both Code of Ethics, one area of concern is the repurposed digital bathroom scale. The scale could potentially violate proprietary information / intellectual property right of Terraillon in the

repurposing manner Team 6 employed. In order to avoid this potential concern, future iterations past the proof-of-concept stage will incorporate a platform and load cell configuration designed by Team 6.

5.7 Future Progress

The KOS only requires troubleshooting in order to obtain a working prototype. Troubleshooting encompasses correcting the Python coding syntax, getting the Pi to register inputs from the GPIO pins, and building a correct circuit to bridge the bathroom scale and Arduino. After a functional prototype milestone is achieved, field testing will be performed, where an evaluation of the effectiveness will be tested though different trials:

a. Populating Template Database trial: Team 6 headquarters is located in Newton, MA, where there are numerous rabbits that ~~host parties~~ emerge overnight. This gives Team 6 an excellent test subject, starting with saving images in the event of motion. Team 6 could then go through and sort the results for positive matches, and incorporate those as template images.
b. KOS trial: After populating the template database, a real test in the effectiveness of our design could occur. The KOS will be subjected to another field test with Scale and Vision structures activated. In the morning a thorough examination at the success rate and a documentation of the battery consumption can occur.
c. Refinement trial: Finally, based on the KOS trials, adjustments will be made to the design, and final trials on the effectiveness of the KOS can occur.

Additions to the design, such as a small HDMI display for researchers to check the recent images and battery life are also in consideration once the KOS passes the field testing trials. Another possible addition includes the introduction of a WiFi USB module onto the Pi, opening up possibilities for wireless communication, monitoring, and data transfer.

6. Conclusion

The Kiwi Observation System is a cost-efficient, field-capable design that will allow researchers to obtain image data of the little spotted kiwi species. Of the Vision structure, multiple object recognition methods were evaluated, resulting in the selection of template based keypoint matching. The Scale structure remains in an infant state due to the time constraint. The design, however, is a few steps away from becoming a functional prototype. With the completion of a prototype, the KOS can then be evaluated for possible enhancements as identified in Future Progress. In summary, the KOS' core concepts will allow this design to accomplish the objectives of object recognition and weight measurement, resulting in accurate data for the research lab.

References

[1] Jolly, J., 1989, “A Field Study of the Breeding Biology of the Little Spotted Kiwi (Apteryx Owenii) with Emphasis on the Causes of Nest Failures,” Journal of the Royal Society of New Zealand, 19(4), pp. 433-434.

[2] BirdLife International 2012, 2012, “Apteryx Owenii,” IUCN 2012, 2012(2).

[3] Holzapfel, S., Robertson, H., McLennan, W., Hackwell, K., Imprey, M., 2008, “Kiwi (Apteryx spp.) Recovery Plan 2008-2018,” Threatened Species Recovery Plan 60, NZ Department of Conservation.

[4] Martin, G., Wilson, K., Wild, M., Parsons, S., Kubke, F., Corfield, J., 2007, “Kiwi Forego Vision in the Guidance of Their Nocturnal Activities,” PLoS ONE, 2(2).

[5] Robertson, H., 2004, “Research and Monitoring Plan for the Kiwi Sanctuaries,” Science for Conservation, 241, pp. 18-20.

[6] Demaagd, K., Oliver, A., Oostendorp, N., Scott, K., 2012, “Practical Computer Vision with SimpleCV,” Ingenuitas, Sebastopol, CA, Chap. 1.

[7] Viola, P., Jones, M., 2001, “Rapid Object Detection using a Boosted Cascade of Simple Features,” IEEE Computer Society Conference, 1.

[8] Han, S., Han, Y., Hahn, H., 2009, “Vehicle Detection Method Using Haar-Like Feature on Real Time System,” World Academy of Science, Engineering and Technology, 59.

[9] Giralt, AS., 2001, “On texture description,” Computer Vision Center, pp. 8-18.

[10] Triggs, B., 2004, “Detecting Keypoints with Stable Position, Orientation, and Scale under Illumination Changes,” European Conference on Computer Vision, pp. 1-3.

[11] “An introduction to load cells, history, theory and best operating principles,” Omega Engineering, from http://www.omega.co.uk/prodinfo/load-cells.html

[12] Horowitz, P., Hill, W., 1989, “The Art of Electronics,” Cambridge University Press, New York, NY, Chap. 7, 15, pp. 421-428, 1001-1002.

[13] 2010, "Arduino Uno R3," Arduino, from http://arduino.cc/en/Main/arduinoBoardUno

[14] Scherz, P., Monk, S., 2013, “Practical Electronics for Inventors,” McGraw-Hill Companies, Maidenhead, Chap. 8, 13, pp. 635-648, 900-908.

[15] 2012, “RPi Hardware,” Embedded Linux Wiki, from http://elinux.org/RPi_Hardware

[16] 2012, “Trademark Rules,” Raspberry Pi Foundation, from http://www.raspberrypi.org/trademark-rules

[17] Richardson, M., Wallace, S., 2012, “Getting Started with Raspberry Pi,” O’Reilly Media Inc., Sebastopol, CA, Chap. 2-3.

[18] 2012, “RPi Performance,” Embedded Linux Wiki, from http://elinux.org/RPi_Performance

[19] 2012, “Learn,” Sight Machine, from http://www.simplecv.org/learn/index.html

[20] Robertson, H., Colbourne, R., 2003, “Kiwi (Apteryx spp.) Best Practice Manual,” Department of Conservation, NZ, pp. 63.

[21] Colbourne, R., 1990, “Relationship between invertebrates eaten by the little spotted kiwi, Apteryx owenii, and their availability on Kapiti Island, New Zealand,” New Zealand Journal of Zoology, 17(4), pp. 541.

[22] “Code of Ethics of Engineers,” American Society of Mechanical Engineers, from http://www.asme.org/groups/educational-resources/engineers-solve-problems/code-of-ethics-of-engineers

[23] “IEEE Code of Ethics,” Institute of Electrical and Electronics Engineers, from http://www.ieee.org/about/corporate/governance/p7-8.html

Appendix A: List of Figures

Appendix B: List of Tables

Table 1. Comparison of size between average little spotted kiwi measurements and Mr. Samsa, a plush replica.

Table 2. Bill of Materials for the KOS

Part	Description	Qty	Cost	Source	Total
Raspberry Pi	Model B with Clear Case	1	43.99	MCMelectronics.com Part # 83-14631	43.99
Arduino Uno	R3	1	29.99	Makershed.com Product Code: MPSK11	29.99
Terraillon Bathroom Scale	TX 6000, Digital	1	20.00	Bed, Bath, and Beyond	20.00
Adafruit camera OR Logitech Webcam	Weatherproof TTL Serial Camera OR Logitech C615 webcam	1	55.00	Adafruit.com ID: 613 OR Amazon.com	55.00
Pelican Case	1120, Foam	1	26.24	BHphotovideo.com	26.24
INA125P Instrumentation Amplifier	Texas Instrument integrated circuit	2	6.43	Mouser.com Part No. 595-INA125P	12.86
Raspberry Pi Starter Pack	Kit containing: case, USB cable, power adapter, USB to TTL cable, Cobbler kit (GPIO), USB microSD reader, breadboard, Ethernet cable, badge, resistors, LEDs, pushbuttons, photocell, capacitor	1	69.95	Adafruit.com ID: 955	69.95
Terminal Block	3-pin, 3.5mm, 5-pack	1	3.95	Adaruit.com ID: 725	3.95
HDMI cable	AmazonBasics, 6.5 Feet	1	5.79	Amazon.com ASIN: B003L1ZYYM	5.79
Arduino Project Enclosure	Arduino Uno case	1	11.95	Sparkfun.com Part: PRT-10088	11.95
Final Cost: $279.72 USD					

Table 3. Project Total Cost

Type	Description	Qty	Cost	Total
Compensation	Total of 200 hours per resource	4	30.00/hr per resource	24,000.00
Materials	See BoM	-	279.72	279.72
Final Cost: $24,279.72 USD				

Appendix C: List of Equations

Equation 1: Analog to digital conversion that allows microcontrollers to receive analog input and convert it into a digital output that can be manipulated. In our case the analog input is a voltage. Since the Arduino Uno's ADC is 10-bits we use:

$$\mathrm{b^n} - 1 = \mathrm{max_{10}}$$

Where,
b^n = Base length
max_{10} = Maximum steps for base 10

Making a 10-bit ADC:

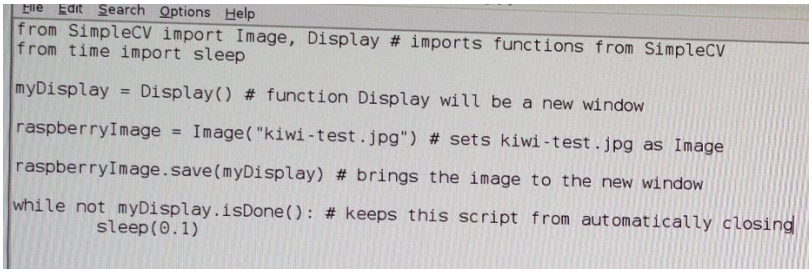
$$2^{10} - 1 = \ 1024_{10} = 1024 \text{ steps}$$

Taking this 1024 steps, we then explore Arduino's limitation of 5V using:

$$\frac{5\text{V}}{1024 \text{ steps}} = \ 0.0049\text{V per step} = 4.9\text{mV per step}$$

Appendix D: List of Code

Code 1: Initial examination of SimpleCV's operation. This code brings up an image and puts it onto a display.



```
File Edit Search Options Help
from SimpleCV import Image, Display # imports functions from SimpleCV
from time import sleep

myDisplay = Display() # function Display will be a new window

raspberryImage = Image("kiwi-test.jpg") # sets kiwi-test.jpg as Image

raspberryImage.save(myDisplay) # brings the image to the new window

while not myDisplay.isDone(): # keeps this script from automatically closing
        sleep(0.1)
```

Code 2: Creation of an application that will save an image of the desktop.

```
[Desktop Entry]
Type=Application
# Choice of Link, Directory, or the chosen.

Name=Screenie

Comment=Screenshots
# Tooltip in case I forget.

Categories=Screen Print
# Menu categorization

Exec=/home/pi/bin/Screenie.sh %U
# Executes the crap in Screenie.sh

StartupNotify=true
# Tracks app startup and provides feedback.

Terminal=false
# Keeps it from crying home to LXTerminal
```

Code 3: Testing a Logitech webcam using SimpleCV.

```
from SimpleCV import Camera, Display
from time import sleep # Something I dearly miss.

myCamera = Camera(prop_set={'width': 640, 'height': 480})
myDisplay = Display(resolution=(640, 480))
# Creates a 640 by 480 window for the webcam

while not myDisplay.isDone():
        myCamera.getImage().save(myDisplay)
        sleep(.1)
# Displays an updated picture from the webcam
# and refreshes for a new image every 0.1 seconds
```

Code 4: Haar-like features using a human face cascade. The application calls you a ninja if it doesn't detect a face. Otherwise, it calls you ugly.

```
from SimpleCV import Camera, Display
from time import sleep # Something I dearly miss.

myCamera = Camera(prop_set={'width': 320, 'height': 240})
myDisplay = Display(resolution=(320, 240))
# Creates a 640 by 480 window for the webcam
# Resolution lowered to 320 by 240 to speed things up

while not myDisplay.isDone():
        frame = myCamera.getImage()
        faces = frame.findHaarFeatures('face')
# Seeks for faces as defined by Haar algorithm, and sets the frame as a face object.

        if faces:
            for face in faces:
                print "I sees you! Youse an ugly one, neh?"
        else:
            print "Are youse a ninja?"
# If/Else statement with a response for positive face and another for negative face.

        frame.save(myDisplay)
        sleep(.1)
# Displays an updated picture from the webcam
# and refreshes every 0.1 seconds
```

Code 5: Template based keypoint matching. This application outlines matches with green rectangles.

```
from SimpleCV import *
from time import sleep

samsa_side = Image("/home/pi/kos/mrsamsa_side.jpg")
side_temp = Image("/home/pi/kos/side_template.jpg")
side_temp_head = Image("/home/pi/kos/side_head_template.jpg")

side_match = samsa_side.findKeypointMatch(side_temp)
side_head_match = samsa_side.findKeypointMatch(side_temp_head)

side_match.draw(width=3)
side_head_match.draw(width=3)
samsa_side.show()
sleep(10)
```

Code 6: Preliminary trial of a motion detection application. The plan is to use this for an application that will populate a folder, allowing researchers to select template images.

```
from SimpleCV import *

webcam = Camera(prop_set={'width': 640, 'height': 480})
threshold = 5.0

while True:
        prev = webcam.getImage()
        time.sleep(0.5)
        curr = webcam.getImage()
        diff = curr - prev
        matrix = diff.getNumpy()
        mean = matrix.mean()

        diff.show()
        if mean >= threshold:
                print "Contact! 12 o'clock, 5 meters, 1 flightless bird!"
```

Code 7: Current progress for a functional KOS Cam application, with various notes.

```
# What this is trying to do:
# I left click, webcam takes a picture (or maybe it turns the webcam on and snaps a picture)
# The image is then processed through keypoint based template matching
# If any keypoints match the 9 templates, the image is saved in directory /home/pi/kos/koscam
# Whether an image matches or not, the camera will wait for another left click

# Need to make a class / object / whatever python calls it to support all of templates that
# KOS Cam is scanning against (templateList, frownieFaces,WwhoCaresStuff)

# you take an image, compare it against everything in that list, if you find any hits, you save
# log useful stuff along the way so it's easy to run this
# Possibly put it in a loop that goes forever or until you press control-c, and sleeps every 5
# seconds, when you hook it up to a webcam

from SimpleCV import *
import time

# this is an image we used to test the template matching.
# It should really come from the camera...
# ...AKA webcam = Camera.(prop_set={'width': 640, 'height": 480"}
scanned_img = Image("/home/pi/kos/mrsamsa_side.jpg")

# side-based templates
# Make a list of all the templates, where they all are templateList[index].name = "Name you can
# message", templateList[index].Image = Image("/home/etc"
side_temp = Image("/home/pi/kos/side_template.jpg")
side_head_template = Image("/home/pi/kos/side_head_template.jpg")
side_claws_template = Image("/home/pi/kos/side_claws_template.jpg")

# front-based templates
front_temp = Image("/home/pi/kos/front_template.jpg")
front_head_template = Image("/home/pi/kos/front_head_template.jpg")
front_claws_template = Image("/home/pi/kos/front_claws_template.jpg")
# ground-based templates
ground_temp = Image("/home/pi/kos/ground_template.jpg")
ground_head_template = Image("/home/pi/kos/ground_head_template.jpg")
ground_claws_template = Image("/home/pi/kos/ground_claws_template.jpg")
# checks the side-based templates
match_side = samsa.findKeypointMatch(side_temp)
match_side_head = samsa.findKeypointMatch(side_head_template)
match_side_claws = samsa.findKeypointMatch(side_claws_template)
# checks the ground-based templates
match_ground = samsa.findKeypointMatch(ground_temp)
match__ground_head = samsa.findKeypointMatch(ground_head_template)
match__ground_claws = samsa.findKeypointMatch(ground_claws_template)
# checks the front-based templates
match_front = samsa.findKeypointMatch(front_temp)
match__front_head = samsa.findKeypointMatch(front_head_template)
match_front_claws = samsa.findKeypointMatch(front_claws_template)
```

```
# Draws a rectangle around matches
# T this turns in to something like
int matchedImages = 0;
foreach(template in templateList)
{
	try
	{
		match = referenceImage.findKeypointMatches(template.Image);
		if(match)
		{
			Log(“Found a match on: “ + template.Message);
			matchedImages++;
			match.draw(width = 3);
			# Does this actually modify the referenceImage?
	# Or maybe something like
			referenceImage.DrawMatch(template.Image)
		}
	}
	catch
	{
		Log(“Looks like no match on: “ + template.Message”);
	}
}
Log(“I found this many matches: “ +matchedImages);
referenceImage.show()


if(matchedImages > 0)
{
	referenceImage.Save(somewhere)
}

match_side.draw(width=3)
match_side_head.draw(width=3)
match_side_claws.draw(width=3)
samsa.show()

# If this whole thing is in a loop that’s always running, then you sleep 10 (seconds?
milliseconds?) and keep taking pictures... cool
time.sleep(10)

#save the image to /kos/koscam if the image matches any of the 9 templates.
```