Raspberry Pi, Python, Digital Cameras, and Speed Detection: Lessons Learned

Emily Minch

Abstract – The purpose of this project was to develop a speed detection camera for a residential user. A Raspberry Pi processor and a surveillance/security camera were chosen as the hardware for this project. The Raspberry takes several images of the passing car, calculates the vehicle's speed, and stores the data and pictures on its memory card or an external drive. This data log could then be taken to police to show that there is a speeding problem in the neighborhood. Image thresholding is used as the method for speed detection rather than RADAR due to its greater reliability, especially in certain weather conditions. Using this camera will not compete or interfere with the efforts of the local police and it can be helpful in solving a speeding problem in the user's neighborhood.

Keywords: Raspberry Pi, microprocessor, computing, image thresholding, speed detection

Project Goals and Materials

Speeding is ubiquitous in today's society, especially in or around residential areas. Lower speed limits in these areas do almost nothing to deter speeders, and their law-breaking can put property, pets, and children in danger. This project aims to provide a cheap and simple surveillance and calculation system so that residents can take frustration with speeders into their own hands. This system does not try to replace local law enforcement, but rather provides a log with images of speeders that residents can take to the local authorities as evidence of a speeding problem in the neighborhood.

The materials for this project include a Raspberry Pi Model B Revision 2.0 processor, Logitech USB webcam, USB mouse/keyboard, FOSCAM FI8904W wireless IP camera and included materials, ethernet access, external display (19-inch Vizio television), 16 GB Sandisk SD card, ImageJ processing software, and HDMI cable.

Introduction

Methods of Speed Detection

The two primary methods for detecting a vehicle's speed are radio detection and ranging (RADAR) and image thresholding/processing. RADAR is the method that police use to determine a vehicle's speed, and such systems can be several hundred dollars. RADAR guns use a concept called a Doppler Shift, which is a change in pitch due to relative motion between the source and the observer, to measure the speed of an object. The RADAR gun can detect how much the frequency of the moving object is changing; therefore, it can show how quickly an object is moving toward or away from it. However, RADAR comes with many disadvantages. Since RADAR has been used for speed detection for over 50 years, many ways to interfere with the signal have been developed. For example, consumers can purchase detectors for their car that will alert them when RADAR is being used in the area or signal scramblers that will distort the signal being sent to the detector. Such devices are illegal in many states, but many drivers use them anyway in order to avoid ticketing. Additionally, angled rain and rough weather can negatively affect the RADAR method, so readings during inclement weather would have to be thrown out.

Image processing uses a simple three-step algorithm to determine the speed of a moving object. The first step in the process is image detection. Image detection is accomplished through thresholding, which is essentially defining a

grayscale cutoff point and categorizing each pixel as black or white based on its location relative to the grayscale cutoff. Thus, each pixel will be either black or white. The resulting image shows the object of interest in the frame so that it can be easily manipulated (Figures 1 and 2). The user chooses which color is the foreground and which is the background. White is the foreground in Figure 2.

Figures 1 and 2. Original image and image after thresholding is applied

The step after object detection is object tracking. As in the images above, the parked cars are not of interest. Using background subtraction will determine whether or not an object is moving over several frames (Figure 3). Subtracting the background of two frames will take out all of the pixels that do not change, resulting in an image of the object that is moving. The last step in the process is speed calculation. In Figure 2, the leading edge of the moving object is used in speed detection. Using a scale in the images will determine how far the leading edge of the object moved versus how long it took for that to happen. A rudimentary calculation of rate as the quotient of distance and time can then be used.

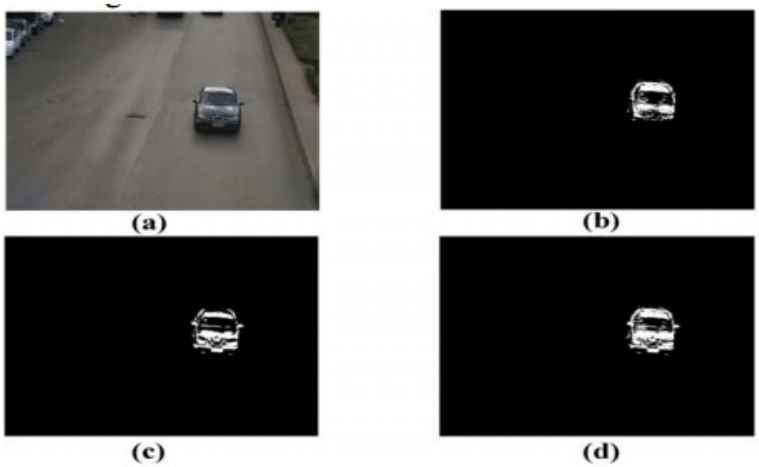


Figure 3. Illustration of background subtraction

Raspberry Pi

The Raspberry Pi is a single-board processor about the size of a credit card. It was developed by the Raspberry Pi Foundation in the UK mainly for the purpose of teaching basic computer science in schools. The Pi's hardware includes 2 USB ports, HDMI and Ethernet hookups, SD card slot, memory, video/audio outputs, and power source. The Pi runs a Linux operating system and recognizes Python as its programming language.

There are several other microprocessors that are similar to the Raspberry Pi. The other options considered for this project were the Arduino Uno and the BeagleBone, manufactured by BeagleBoard. The BeagleBone and the Pi seemed to be the most relevant to the project, but the Pi is a cheaper option and has more of a community around it for embedded systems projects. Additionally, the makers of Raspberry Pi are working on a camera module that will be highly compatible with the board itself. The camera module, however, was not yet released at the time of testing. Since the camera module was not available for use, the camera had to be selected from a list of hardware that is compatible with Linux.

Since the Pi is a very basic computer, it requires an external display connected with an HDMI cable (a TV for this project), a USB keyboard and mouse, and an ethernet cable to connect to the internet. A USB hub is also helpful since the Pi only comes with two USB ports. As soon as all of the hardware is assembled, there are several steps to go through in order to boot the Pi.

Image detection and processing with the Raspberry Pi could be done one of two ways: Python programming or the command terminal. There are several online articles about using Python for image processing on the Raspberry Pi, but importing the Python libraries proved to be more difficult than it seemed. As an alternative, a user can import several pre-written libraries through the command terminal. One such library that was useful for this project is the motion library. The many settings for motion can be edited in the configuration files using the sudo nano /etc/motion/motion.conf command, and motion is started with the sudo motion command. The camera will then take images continuously, for a specified amount of time, or only when it detects a moving object, depending on the setting. Every image will be saved to a temp directory on the Pi, or another location specified by the user. Motion will put an outline around the moving object, but that did not prove to be extremely accurate when taking images of a car. Figure 4 shows that in an image taken of the car, the car is on the left side of the frame while the motion box is on the right. It is a nice feature in theory, but did not translate to the execution phase well. Additional libraries for the Raspberry Pi that are used in conjunction with a webcam are MJPEG streaming and fswebcam. Motion was chosen for this project since it is the most relevant to the project.

PROCESS

The initial plan for this project was to link the wireless IP camera to the Raspberry Pi and to send the data through the shared network. However, due to the initial problems and rudimentary nature of the Pi, it was decided to use a USB webcam plugged directly into the Pi. The Pi communicated with the USB webcam much easier since the hardware was directly linked. The main problems encountered were the lack of public Ethernet ports at Mercer University paired with the University's firewalls. The University's IT department is currently implementing a plan to phase out Ethernet ports in student housing, and the Pi requires an Ethernet connection in order to boot and correctly set up the operating system. Therefore, the system had to be taken to several locations in order to resume work. The problem encountered while attempting to use the wireless IP camera was that the University's firewall did not allow for data sharing between the processor and the camera while they were using the University's connection. Firewalls had to be bypassed for the camera to initially be set up, and it quickly became clear that communication over the network was not a realistic expectation. Since the USB webcam was physically connected to the processor, it did not need to communicate through the network. Although that was a large hardware downgrade, it saved time, effort, and potential consequences for violating the University's network usage policies.

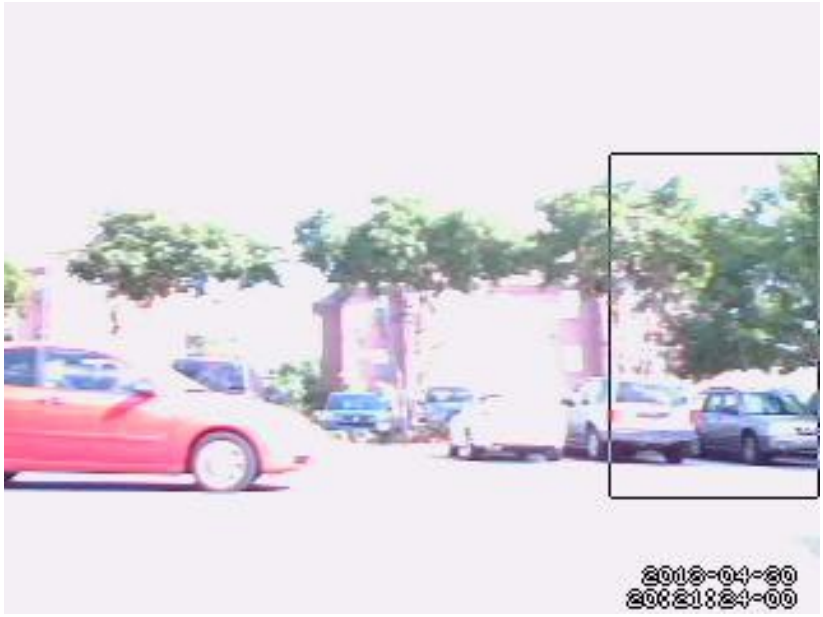
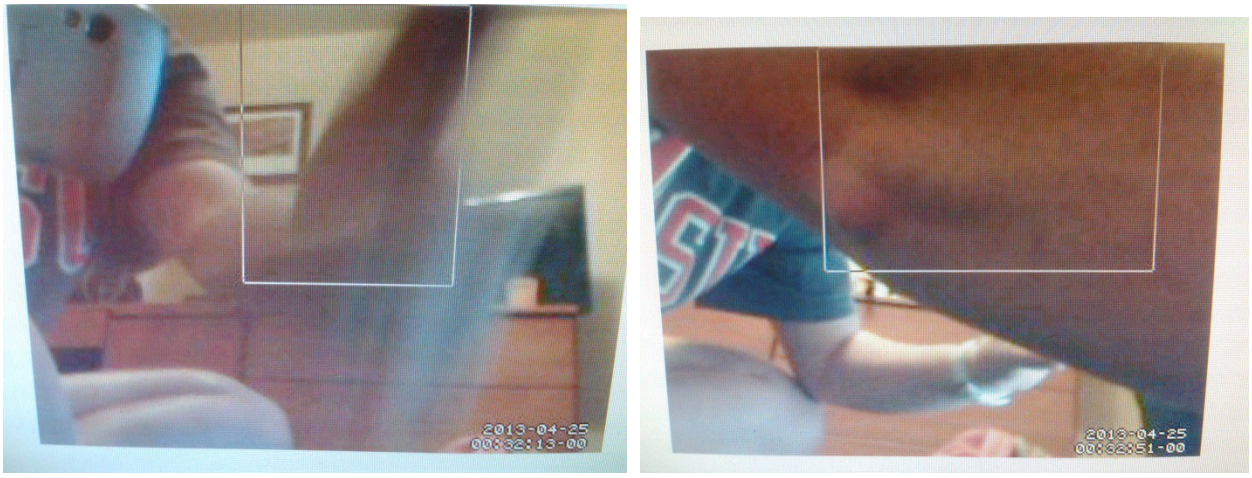


Figure 4. Example of incorrect placement of motion box.

After making the switch to a USB webcam and getting the motion libraries to work through the command terminal, tracking a moving object could commence. Even after several initial tests, the camera and processor had trouble communicating. The camera would turn on and capture images for one test, but not turn on at all for several subsequent tests. The motion libraries were uninstalled and reinstalled several times until the camera was working semi-consistently. The initial test was simply waving a hand back and forth in front of the camera to ensure that the camera was correctly tracking and documenting a moving object. The pictures looked promising: there was a box around the moving object, the arm, and a (seemingly) correct timestamp in the bottom right corner of each picture (Figures 5 and 6).



Figures 5 and 6. Sample images from the original motion tests

Simple image processing and calculations were performed on the arm images to test the method for determining speed. However, since the true speed of the arm was not known, the timestamp and frame number between the two

images could not be tested to make sure that it was accurate. Thus, an object that had a known speed needed to be tested next: a car.

The frame rate was set to 50 frames per second to ensure that a large number of good images were taken. Theoretically, the frame number should have started at 00 and ended at 49 for each second. In initial tests, before the camera was working correctly, the system "captured" and saved hundreds of trash images in the designated folder. These images did not show anything except a gray background, but they had the date and time information in the bottom corner. Analyzing those showed that the actual frame rate ranged from 9 to 25 frames per second, and each second had a different number of frames. None of the sets of images reached 50 frames, so it was obvious that the system could not handle that high of a frame rate. The experimental frame rate was then lowered to 20 frames per second. After moving the entire project apparatus outside, a truck was driven around the parking lot twice at a fixed speed. Unfortunately, the Pi did not capture the truck driving by either time. It is possible that the camera stopped capturing images for several seconds and started again after the truck had already passed. All terminal coding was done blindly, since it was not feasible to carry the entire apparatus (including the TV) outside. The Pi was being powered through a USB port on a laptop, but the TV required a traditional 3-prong outlet that was not present outside. Therefore, there was no monitor to see what code was being typed into the terminal and it was all done from memory. Fortunately, the Pi perfectly captured a red car that happened to drive by (Figure 7). Although this made it difficult to know the exact speed of the passing car, the speed could be estimated based on the control speed and the fact that it was tested in a small parking lot. The speed of the car was estimated to be 17.5 mph, which is halfway between 15 and 20 mph. The control speed was 20 mph, and the red car was driving behind the control car. Therefore, due to the physical limitations of the parking lot and the control car, the speed estimation is reasonable. Figure 7 shows that the images from outside turned out to be very light. The brightness of the images interfered with the image thresholding and made it harder to analyze the images.

Figure 7. Sample shot of red car moving past the camera.

Before the speed of the car could be measured through image processing, the images had to be scaled correctly. The viewing window of the webcam spanned six parking spots, so one of them was measured with a tape measure and multiplied by six. The experimental speed of the car could then be calculated and the theoretical speed of the car could be used to test the accuracy of the time stamp.

The first step in this process was to measure the distance that the car moved between frames, the images had to be scaled to the real-life distance. Since the length of the six parking spots was known, the scale was determined by the length of the entire frame. Two consecutive images were processed and the length that the car moved in between the

two frames was measured. Six usable images were captured and the test was performed on more than one set of images.

The first test did not yield ideal results. The car was calculated to be going 25.6 mph- a practical speed, but not for the testing environment. Tests on the other frames showed similar speeds. All speeds were approximately 24 mph, which is slightly high for the testing environment and more than 7 mph the theoretical speed. At that point, it became obvious that the timestamp on the photos was incorrect. Many calculations proceeded in order to determine the lack of accuracy on the timestamp.

The frame rate was set to 20 frames per second. Therefore, theoretically there should have been 20 images for each second numbered 00 to 19 at the bottom. The six frames captured were numbered 02-04 for one second and 00-02 for the following second, so it appeared that the number of frames for the first second was only 5. The calculation of the average number of frames using the theoretical speed came out to 3 frames per second, almost 1/7th of the theoretical.

Conclusions and Future Recommendations

The main conclusion from the tests in this project is that the software could handle the high theoretical frame rate, but the hardware could not. The motion libraries showed that the frame rate could be very high, but it did not work during testing. This was to be expected considering the materials that were used for this project. The microprocessor is made to be used for educational projects, and the camera was a simple and cheap USB webcam. The system of the webcam and processor could not take 50 pictures in one second. A more expensive IP camera was originally purchased for use with this project, but it was very incompatible with the Pi and would have taken up much of the testing time in order to work properly.

This project originally had ambitious goals to be a fully working system that could be regularly tested in the ideal environment, but the myriad problems with the testing inhibited the final outcome of the project. The Python image processing libraries would not import correctly, so the thresholding and calculating was done manually rather than by the Pi. This project could be immensely helpful in the future if the entire system could become automated. The appropriate equipment could be upgraded, although this would increase the cost of the system. The ability to capture and store pictures in a log was demonstrated as was the potential for estimating speed using the thresholding technique. This project embodies the “getting what is paid for” principle in that inexpensive equipment will not always yield ideal results.

Image processing is an advanced programming task, and more than just basic knowledge of Python is recommended for anyone attempting to further this projects efforts. Troubleshooting was very tedious since there was not an in-depth knowledge of Python programming, and the errors were not easily recognizable as either software or hardware related.

A hidden, automated speed monitoring system that could be located outside a citizen’s home and generate a log and images of speeders in the neighborhood still has many potential benefits as well as consumers. This project has identified a variety of software and hardware challenges that must be addressed by future endeavors. Embedded systems projects are a wonderful teaching tool as it ties in both the hardware and software components of computer engineering. There is a large need for embedded systems engineers, and this project was wonderful experience in that field.

Acknowledgements

This work would not have been possible without the financial and administrative support of the Engineering Honors Program at Mercer University. The author would like to thank Dr. Philip T. McCreanor, Director of the Engineering Honors Program, for his guidance in developing, implementing, and documenting this project. The author additionally thanks Dr. Anthony Choi, Assistant Professor in the Electrical and Computer Engineering Department, for his guidance on embedded systems projects.

References

[1] Osman Ibrahim, Hazem ElGendy, and Ahmed M. ElShafee, "Speed Detection Camera System using Image Processing Techniques on Video Streams," *International Journal of Computer and Electrical Engineering* vol. 3, no. 6, pp. 771-778, December 2011.
[2] Sato, Y., "Radar Speed Monitoring System," *Vehicle Navigation and Information Systems Conference*, 1994. Proceedings., pp.89-93, Sep 1994.

Emily Minch

The author is a senior at Mercer University pursuing a Bachelor of Science in Engineering with a computer specialization, as well as minors in Mathematics and Technical Communication. Her coursework has included electrical and computer hardware courses as well as software courses through the Computer Science department. She has completed many research projects through her four years in the Engineering Honors Program as well as the Freshman and Senior Engineering Design courses at Mercer University's School of Engineering. Her research projects have featured image thresholding, Second Life® programming, and GPS waypoint navigation for unmanned aerial vehicles.