Matthew Flaherty
Jackson Foster
Alexander Jiao

Raspberry Pi Motion Tracker Final Report

Introduction

This project implements an object motion tracker using a Raspberry Pi 3 (RPi) and OpenCV. The RPi is a low-cost and portable computer that contains both a Linux distro (Raspbian) and low-level I/O ports. It can also be paired with a low-cost camera module, making it suitable for this project. OpenCV, or the Open Source Computer Vision Library, is an open source Python/C++ library that contains a wide range of computer vision and machine learning functionality. It's extensive usage by both hobbyists and professionals as well as the provided documentation made OpenCV the best choice for our project.

Ultimately a motion tracking device is created that allows a person to be selected, and then follows that person through the camera's range of motion. This report will discuss the technical details and choices of the implementation.

Functional Overview

The tracker consists of a RPi that controls two servos which rotate a camera on two axes in order to track objects moving either horizontally or vertically across the camera's field of view. Mechanical details of the setup will be discussed later. Figure 1 shows the software diagram of our system.

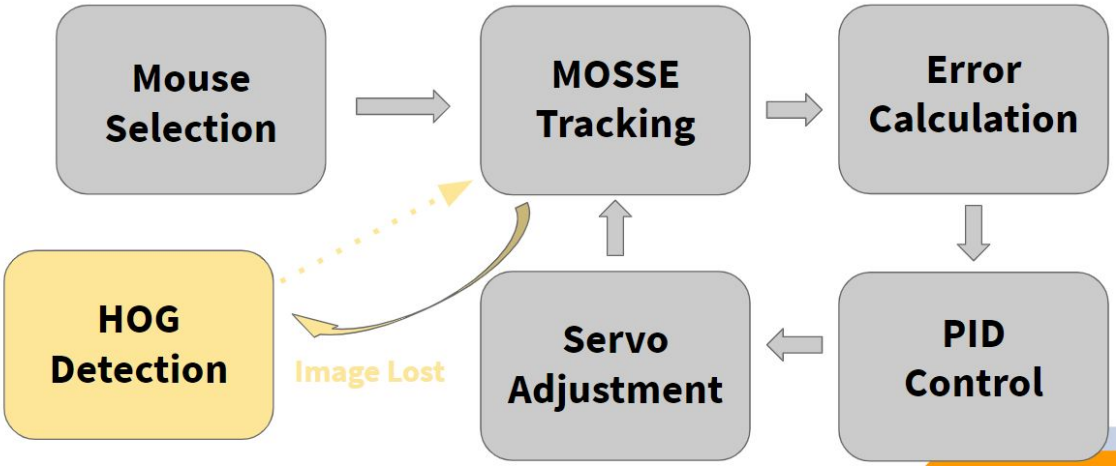


Figure 1. Software system overview

At first, a GUI will appear that displays the camera feed and allows a user to select a region of interest (ROI) on the screen in the form of a bounding box. Then this ROI is passed to a Minimum Output Sum of Squared Errors (MOSSE) object tracking algorithm which updates the bounding box of the object on successive frames from the camera. This passage of the bounding box from the HOG to MOSSE tracker is based on a source by Rosenbrock using a similar methodology [1]. By using the distance from the center of the bounding box to the center of the camera as an error, a PID control loop is updated which then adjusts the servo to track the object. If at any point the object is lost or is stationary for too long (indicating that MOSSE may have accidentally started tracking an inanimate object), a Histogram of Oriented Gradients (HOG) technique can be used to redraw a bounding box around a person on the screen.

Object Identification

The object identification we chose to use was a Histogram of Oriented Gradients (HOG) approach with a linear Support Vector Machine (SVM) classifier. This technique was suitable because pre-trained models could be loaded onto the RPi which saved us the costly operation of finding training data to tune our own models.

HOG is a method of extracting descriptors from images first introduced by Dalal and Triggs in [2]. In HOG, the region of interest (often a subset of the entire image) is broken into smaller cells of size 4x4 to 12x12 pixels. The gradient at each pixel in each cell is computed. Many methods of taking the pixel gradient exist but in general computing the difference between right and left pixels suffices for the x-gradient and the difference between top and bottom pixels suffices for the y-gradient. The x and y components are then used to compute the total gradient magnitude and direction at that pixel. A histogram of gradients is then generated for each cell by binning each gradient by its direction and adding its magnitude to the frequency count. In this way a 1-D histogram is created for each cell, and the histograms for all cells can be combined to form a feature vector for the entire region of cells.

Support Vector Machines are a common method for performing binary classification of data. The general theory of SVMs is well developed and will not be discussed in detail here; more information can be found in [3]. The basic idea of an SVM is to draw a boundary between two classes of data in high-dimensional space by maximizing the total margin between the boundary and the data points of each class that sit closest to the boundary. Our model is a linear SVM which means the boundary is a hyperplane. The SVM is trained on images containing people and containing no people.

OpenCV contains a pre-trained person detector object that uses HOG and SVM as described above, but the classifier could also be trained manually. The framework for

training the SVM and then finding a bounding box on a person in a new image is as follows. Steps taken from [4].

1. Gather images containing people, and images containing no people for training. There should be many more images containing no people. Extract HOG feature vectors from both the positive and negative training sets.
2. Train the linear SVM on the training image feature vectors. This can be done with the popular Scikit-Learn or LibSVM.
3. Perform hard-negative mining. Scale each image of the negative training set, apply a sliding window to the image and classify each window using the SVM. Save the feature vectors of negative images that were classified as having people. Do this for multiple scales.
4. Retrain the classifier using the false-positive samples from the previous step.
5. For any new images that need to be classified, apply the same scaling then sliding-window approach. Any window that is classified as having a person is taken to be the bounding box of that person.

Due to the sliding window approach, many bounding boxes are sometimes generated for a single person. A technique called non-max suppression (NMS) can be applied to remove the extraneous boxes. In NMS, each box is checked to see if it overlaps with other bounding boxes. If the proportion between the overlapping area and the area of the smaller box reaches a user-determined threshold then the smaller box is removed. This technique is described in greater detail in [5].

For our purposes we took the largest box in the image to be our bounding box. This was for two reasons: we wanted to avoid the extra computation time associated with NMS and we could only track one person with the camera anyways. The OpenCV HOG SVM implementation also returns a weight corresponding to how likely the object in the box is human; taking the box with the highest weight is another viable alternative.

Since HOG is too expensive to compute on every frame from the camera feed, we only compute it when certain criteria are met. If the object tracker algorithm loses the object it was supposed to check for a set number of frames, then HOG is used to find a new bounding box on a person in the camera's frame of view. This bounding box is used to re-initialize the tracker. Additionally, if the tracked object remains stationary for a set number of frames then it is likely that the object tracker has accidentally locked on to a non-human stationary object. In this case HOG is once again used to determine if a person exists in the frame and to return the bounding box of that human.

Object Tracking

After pursuing object identification and finding that object identification would be too computationally intensive for the Raspberry Pi, the team decided to pursue other avenues for tracking objects. Object tracking algorithms require many fewer computations that object identification since they have more information about the object being tracked than object identification algorithms. The initial object tracking algorithm the team implemented was using the Kernelized Correlation Filter (KCF) method implemented in OpenCV. Using this method, we were not able to obtain better performance than for that of the HOG detection ran on every frame.

Minimum Output Sum of Square Errors (MOSSE) was found experimentally to have the best performance. MOSSE is the primal case of the KCF method where there is only one channel considered, the raw pixel values. KCF is able to sum over multiple channels and include features obtained from HOG, however computing these features will still require a computation of HOG on every frame which will take many computations. So while the KCF can include more features in one algorithm, summing over multiple channels, to obtain these features would still require identifying HOG features and therefore the benefits of KCF are lost. Thus we had to run the KCF in its base form using only the raw pixel values as features which reduces to the MOSSE object tracking algorithm [6].

While running the MOSSE tracking algorithm at the original resolution, the team was able to achieve a frame rate of roughly 10 frames per second. This is fast enough to track objects moving at reasonable speeds. When considering the operations involved in running the MOSSE algorithm, it is seen from a paper by Bolme et al that the longest computational element of the algorithm is computing the Fast Fourier Transform and Inverse Fast Fourier Transform of the image [7]. Since the Fast Fourier Transform algorithm can be sped up by operating on matrices whose size is a power of 2, the team resized the image to be 1024 pixels in width by 512 pixels in height. This increased the speed in computation considerably allowing for the object tracking algorithm to run at approximately 23 frames per second.

A strength of the MOSSE tracking algorithm are its ability to change its filter used for computing the new location of an object with every new object frame obtained. Using a learning coefficient, one can weight how highly previous versions of the filter will be added to current versions of the filter. This means that the filter will be able to adapt to small changes in object shape or lighting since these small changes will still highly correlate with the current filter as the object moves through the frame. But the filter will rapidly adapt to the changes in the object. The only case where this does not work well is in the case where the target rotates their body. The filter attempts to remain on the

center of the individual, however the individual has now become thinner. Once the target takes up less than 50% of the bounding tracking box, the MOSSE algorithm tends to lose them since the stationary background is now the majority of the filter and will correlate more highly than the skinnier rotated individual. In this case, the individual will then just walk out of the tracking region while the box remains stationary. However, the adaptivity of the filter works for most other conditions such as the change of shape of the individual walking or moving within the frame. Hence, this adaptivity is generally a strength for the algorithm.

One way to mitigate the pitfalls of the adaptive filter would be to employ background subtraction to remove features from the background which are interacting with the filter. There were several problems with employing background subtraction for our specific application. First, background subtraction requires multiple frames to initiate the background. This would require the camera to pause for several frames and remain still to set a background. This could be done at the beginning of the video and would not be a problem for a stationary camera application. The application for our device involves rotation of the image which would move the entire background of the image as the camera rotates. This would require an algorithm to be written which initiates the camera over many initial frames at each possible camera angle. This background subtraction method would likely also be prone to errors since slight physical deviations of the servo from its assigned angle would cause the entire background of the image to not be subtracted. Since there were many technical challenges involved with the implementation of background subtraction on a rotating camera platform, it was not pursued further in the interest of time.

The adaptive filter also caused several problems for this specific application of a moving camera. Since the camera is feeding errors into the PID controller used to move the servos, larger errors mean large changes in servo angle. When the servo must snap to a distant angle, the large increase in speed blurs the image. When the image is blurred, the filter adapts to look for blurred images. This blur makes the camera continue to locate the image in a similar location as before, causing the error to remain high and the camera to continue rotation. With continuously high error, the camera continues to move and the filter becomes entirely blurred, until the camera is pointing to one side and has hit its limit in rotation. At that point the object is lost and a new object must be tracked. This problem was mitigated by adding a limit in the amount the servos are allowed to rotate in any single step. This causes the camera to move slightly slower than desirable, however the outcome is better than losing the tracked object entirely and steering the camera far to one side.

After achieving good results in tracking objects identified by the user, the team pursued object identification using a HOG algorithm followed by object tracking using the MOSSE algorithm. This method would allow the algorithm to be entirely

implemented using the Raspberry Pi with no need for user input. In this case, the program would grab an initial frame, identify a target to track (in our case the human whose correlation weight is highest, i.e. the most likely to be a person) and pass the bounding box to the MOSSE algorithm for object tracking. If the tracking algorithm loses track of the object, or the object does not move for some set number of frames, HOG is run again to identify the target. A stationary object is considered not to be a person since people tend to move slightly. If the object is a person it would then be re-identified by the HOG detection algorithm. If the MOSSE target filter was modified or lost track of the object and was now tracking a the background since the object gradually moved out of the bounding frame (usually by rotation) then the HOG needs to be recomputed to find the target again.

When actually implementing the HOG to MOSSE algorithm, the identification to tracking handoff was not achieved at a level the team was comfortable showcasing. Initially, there were problems with the HOG algorithm identifying objects that were far to the side of the image causing the image to blur and objects to be lost. After this problem was fixed, it was found that the HOG was not consistently identifying the same object or even the only human object in the frame. Often, the HOG would misidentify other objects as human such as two table legs placed closely together or other objects whose gradients may be similar to that of a person. It was difficult to use only weights to identify what is a person and what is not based only on the weight returned by the HOG algorithm corresponding to its confidence that it has found a person. Thus the tracker was all over the place and could not consistently identify the only person or even one of multiple people within the frame. This issue meant the team decided to showcase on the MOSSE tracking portion of the project since it worked most consistently and could provide a good demonstration of object tracking and servo adjustments.

Controls

When pursuing a control algorithm for modifying servo angles, the team chose to implement a Proportional Integral Derivative (PID) controller since it is simple to implement, can be tuned manually, and requires little computation. Implementation for the PID controller was inspired by a webpage which demonstrated how to implement a simple PID controller in Python for a Raspberry Pi [8].

The PID controller essentially calculates how much to modify the servo angles given the current and previous errors fed into the controller. The errors that were fed into the controller are the current coordinates of the center of the box to be tracked. The coordinate system used has the origin at the center of the screen which allows for the coordinates of the box to be related to the amount that the cameras must be rotated. Using some scaling factors determined from the field of view of the lens divided by the

number of the pixels in that direction (x or y), pixel coordinates can be converted to degrees that the camera is off from the center of the target. These errors are then fed into the PID loop which corrects the camera angle based on coefficients supplied.

The proportional coefficient in the PID controller scales how much the current error will change the current servo angle. The derivative coefficient scales how much the previous error will affect the current rotation supplied to the servos. The integral coefficient will affect how much the sum of the previous errors will affect the next change in servo angle. To tune the device, initially the group used a heuristic method which involved changing the filter coefficients based off behavior observed in the controller. This worked fairly well, however the team had problems with the controller becoming unstable which would make object tracking impossible. So next the team tried tuning the coefficients using the Ziegler-Nichols method which is easy to implement and has been proven to provide reasonably good results [9]. This method involves setting all filter coefficients to zero and tuning the proportional constant until the controller maintains steady oscillations that do not increase without bound. This ultimate coefficient is then scaled appropriately and determines the proportional, integral, and derivative coefficients. This method also includes the amount of time between the errors when computing the sum to take a true integral. The program was modified to include this timing information and new coefficients were set using the Ziegler-Nichols method.

Good tracking results were observed for small to medium changes in the camera using this method. The team noticed that the camera often did not perform well for large changes necessary for initially placing the object in the frame using this method. However, once the object being tracked was placed at the center of the frame, the controller was very responsive and tracked the objects very well in real time. The issue was determined to be as stated earlier in the object tracking section. The controller was changing the angles too quickly and blurring the camera, changing the adaptive filter used for object tracking. This caused the controller to appear unstable, as it was being fed large errors continuously that it could not correct. A check was added to prevent errors over an empirically determined threshold from being implemented as changes to the servo angle. This prevented the camera from moving too quickly, allowing the image to remain clear enough for object tracking to persist more robustly.

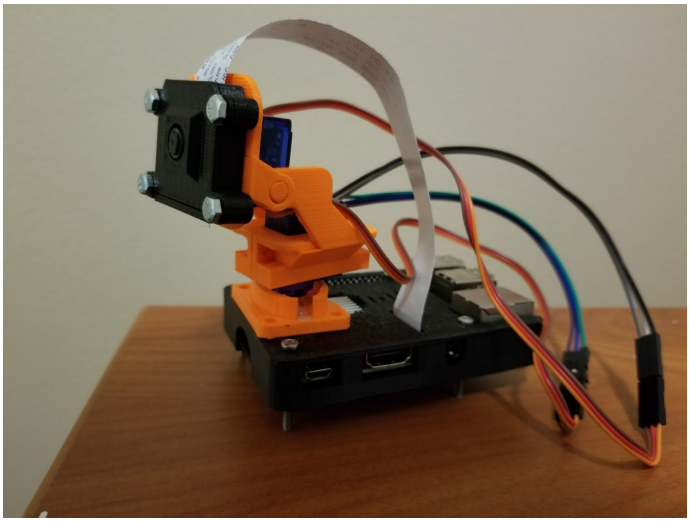
Thus the controls necessary for this project were not necessarily what could be determined optimally from a controls standpoint since the image stabilization was a problem for our camera and servo setup. Improvements could be made by using a camera such as a GoPro which has image stabilization hardware and software allowing the images to remain in focus as the camera is moved quickly. This was unfortunately out of the price range of the group and would have required a large amount of time to get the camera operating within the framework of the Raspberry Pi.

Due to Raspbian being a near-complete Linux distro, it cannot devote full processor attention to driving the servo PWM signals. This is because it must manage many other background processes in addition to the Python script for the object tracker. Because of this, the servo motion was very irregular when the servo control was implemented in a single thread with the motion tracking script. To fix this, the Pigpio package was installed. Pigpio contains a daemon that must be started before the Python script is run; the daemon connects to a C-library that improves the low level I/O of the Pi [10]. During the execution of our program, we first connected to the daemon, then called Pigpio's *set_servo_pwm()* function from our PID loop to control the servos. This fixed the issue.

Gimbal and Case

The camera was mounted to a gimbal mechanism that attached to a Raspberry Pi case, as displayed in Figure 2. The Pi and camera cases were taken from the design by [11] and the gimbal mounting was taken from [12]. All parts were printed in PLA plastic at 20% infill. Two SG90 180° servos were inserted into the gimbal and the entire gimbal mechanism was glued onto the Pi case. The camera housing from [11] was screwed onto the backing plate from [12] in order to hold the camera in place. The only modification to the case itself was using a Dremel to create an opening for a heatsink to fit in with sufficient ventilation. Using this configuration, the camera could be rotated up to 180° in both the θ and φ angles, although we software-limited the ranges of motion to 125° and 70°, respectively, to avoid damaging the ribbon cable or exceeding the limits of the servos.

Figure 2. Raspberry Pi case and gimbal mounting.



Conclusion

The team was able to demonstrate the possibility for object detection and tracking via the Raspberry Pi module. The program has been shown to run at a reasonable frame rate of approximately 23 frames per second and place a moving target within the center of the frame. While there is still room for improvement in object identification, the project showcases the ability of this relatively inexpensive technology to achieve the goal of object tracking using minimal CPU resources. In the future, this project could be improved through purchase of better materials for smoother mechanical operation for stable image capture.

Overall, this project has successfully completed its goal in performing mechanical actuation towards object tracking using the Raspberry Pi 3B and servos available commercially. The applications for this technology are far reaching and include security, sport, scientific observation and more.

References

[1] A. Rosebrock, “OpenCV Object Tracking,” *PyImageSearch*, 30-Jul-2018. .

[2] N. Dalal and B. Triggs, “Histograms of Oriented Gradients for Human Detection,” in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR’05)*, San Diego, CA, USA, 2005, vol. 1, pp. 886–893.

[3] C. J. C. Burges, “A Tutorial on Support Vector Machines for Pattern Recognition,” *Data Min. Knowl. Discov.*, vol. 2, no. 2, pp. 121–167, Jun. 1998.

[4] A. Rosebrock, “Histogram of Oriented Gradients and Object Detection,” *PyImageSearch*, 10-Nov-2014. .

[5] A. Rosebrock, “(Faster) Non-Maximum Suppression in Python,” *PyImageSearch*, 16-Feb-2015. .

[6] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, “High-Speed Tracking with Kernelized Correlation Filters,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 3, pp. 583–596, Mar. 2015.

[7] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, “Visual object tracking using adaptive correlation filters,” in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 2544–2550.

[8] “Going Straight with PID - Introduction | Raspberry Pi Projects.” [Online]. Available: https://projects.raspberrypi.org/en/projects/robotPID. [Accessed: 13-May-2019].

[9] J. G. Ziegler and N. B. Nichols, “Optimum Settings for Automatic Controllers,” *J. Dyn. Syst. Meas. Control*, vol. 115, no. 2B, p. 220, 1993.

[10] “pigpio library.” [Online]. Available: http://abyz.me.uk/rpi/pigpio/python.html. [Accessed: 13-May-2019].

[11] Thingiverse.com, “Raspberry B+ housing, camera housing and servo gimbal for camera by Fido.” [Online]. Available: https://www.thingiverse.com/thing:504196. [Accessed: 13-May-2019].

[12] Thingiverse.com, “SG90 Servo 2-axis Gimbal by aanarcissus.” [Online]. Available: https://www.thingiverse.com/thing:2892903. [Accessed: 13-May-2019].