Development of an Open Source Motion Capture System

Paul Canada
Graphics & Interactive Research Grp.
Western Connecticut State University
Danbury, CT

George Ventura
Graphics & Interactive Research Grp.
Western Connecticut State University
Danbury, CT

Christopher Iossa
Graphics & Interactive Research Grp.
Western Connecticut State University
Danbury, CT

Orquidia Moreno
Graphics & Interactive Research Grp.
Western Connecticut State University
Danbury, CT

William J. Joel
Graphics & Interactive Research Grp.
Western Connecticut State University
Danbury, CT
joelw@wcsu.edu

ABSTRACT

Motion capture (MoCap) has been one of the leading and most useful tools within the field of animation to capture fluid and detailed motion. However, it can be quite expensive for animators, game developers and educators on a tight budgets. By using Raspberry Pi Zeros, with NoIR cameras and IR LED light rings, the cost of a four-camera system can potentially be reduced to less than 1000 USD. The research described should lead to an effective and useful system, able to detect multiple markers, record their coordinates, and keep track of them as they move. With a setup of three or more cameras, one would be able to triangulate the data on a low-cost host computer. All software and hardware designs will be disseminated open source, providing anyone who is interested in MoCap, whether it be for hobbyist, semi-professional, or educational purposes, a system for a fraction of the typical cost.

CCS CONCEPTS

• **Computing methodologies** → **Motion capture**; • **Applied computing** → *Media arts*; • **Hardware** → *Printed circuit boards*;

KEYWORDS

Animation, Motion capture, Single board computers

ACM Reference Format:
Paul Canada, George Ventura, Christopher Iossa, Orquidia Moreno, and William J. Joel. 2018. Development of an Open Source Motion Capture System. In *Proceedings of SIGGRAPH '18 Posters.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3230744.3230811

1 PRIOR ATTEMPTS

Though there are many resources that address motion capture and tracking [Boger 2013], none discuss development using single-board computers such as the Raspberry Pi.[O'Hanlon 2012][Rosebrock 2016][Sharwood 2017] Only one group was found that had begun a similar project. However, they had not posted any progress or have any deadlines. Therefore, this seemed to be the first active attempt at a project of this nature. Since essentially there is no prior work, none to compare against, the author entered this project blind yet still achieved significant progress. At present, a basic, one-camera system that identifies markers and their locations, and roughly tracks said markers has been built.

SIGGRAPH '18 Posters, August 12-16, 2018, Vancouver, BC, Canada
© 2018 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-5817-0/18/08.
https://doi.org/10.1145/3230744.3230811

2 APPROACH

Initially, the thought was to use a Raspberry Pi Zero as the basis for a motion capture camera. However, due to limitations in the Pi Zero, a Raspberry Pi 3 (RPi3) was chosen to begin prototyping. Te RPi3 provides more ports and is easier to use when testing with an IR camera. Also, a version of Linux, Raspbian, was selected as the operating system.

Following these decisions, the next task was to decide which language to program with. Both Processing (a Java based language), and Python were examined as they are both available for Raspbian. Though Processing has the potential of running faster, Python was chosen for its ease of use. It is a language that is often used for prototyping. Python also supports the OpenCV library [Rosebrock 2016][Fac 2017], which has many feature-detection and image-processing functions built in. This eliminated the need to build the code from scratch to detect markers.

3 IMPLEMENTATION AND PROCESS

There are two available distributions of the Linux-based Raspbian OS; Jessie and Jessie Lite. Since some project members were new to or had little experience with a Linux environment, the decision was made to use the full version of Jessie. This reduced the need to spend significant time setting up a desktop environment. At this stage, a basic system was created that would import a small video file and extract frames from the video. Each frame was converted into gray-scale [Ove 2011] and a binary thresholding [Devi 2006][Fisher et al. 2017] was applied on each of the gray-scale frames to filter out different gray-scale levels. Anything above the threshold would appear as white and anything below would appear as black. Te white blobs (markers) would then be the regions of interest.

Initially, a data cable and a camera were connected to the board, and a light ring was mounted on the camera to have it all in place for future stages of the research. However, the camera was not connected to the power ports in the RPi3 GPIO header. This basic setup yielded results as expected. Without the presence of IR light, a low threshold was used to make sure all points of interest were captured. Unfortunately, this also yielded junk data, such as light reflecting of a watch face. After the light ring was powered and emitting IR light, the threshold could be raised to the point where anything that is not a marker was eliminated. Also, a solution to an ongoing capture rate issue was found. It was discovered that the camera interface could be controlled directly with openCV utilizing the Video4Linux driver, as opposed to using the PiCamera module offered by Python/Raspberry Pi. This provided a massive frames per second (FPS) boost.

Once a marker was identified by the camera, a bounding box was drawn around it on the screen. This verified that marker detection was finding the correct regions in each frame. Coordinates were extracted from the bounding boxes to find the center point of each marker. Given the bounding box was displayed near exactly over the marker, a center point was computed near perfect every time. When a marker is determined by the system, an object is created that stores requisite information: coordinate pair (x, y) for the center point, a camera identifier, an identifying number for each marker, and a time stamp.

4 ISSUES SOLVED

4.1 Light Ring Overheating

Initially, after setting up the light ring, the ring itself became extremely hot to the touch, bad for the unit itself, difficult to physically handle. Subsequently, a new light ring was soldered into place with its headers in the opposite direction. This solved the heat issue, indicating that either (a) the first light ring was defective, or (b) the soldering was not sufficient and was causing a short. However, as a safety precaution, a script was created to run on startup that would only enable the light ring when the camera was capturing. This script also helps to reduce power consumption.

4.2 Frame Rate

Also, early on some severe FPS issues arose, where the camera captured at a very low FPS. Te solution was a direct driver for the camera, Video4Linux [10]. When interfacing with the camera via V4L, the FPS increased dramatically. Using PiCamera averaged somewhere between 9-14 FPS at a resolution of 320x240. However, interfacing with V4L yielded around 75-90 FPS at a resolution of 320x240, and 25-40 FPS at a resolution of 640x480. It is suspected that the cause of this is a lack of interfacing with the camera through a Python module, which will be inherently slower than something that is compiled or pre-built. This approach was applied using the RPi3. A FPS of about 15 is expected for the Pi Zero. However, there are still optimizations that can be performed, which will hopefully yield even better results, especially for the Pi Zero.

5 ISSUES REMAINING

5.1 Raspberry Pi Zero

The reason the Pi Zero is priced at 5 USD is because its processor and RAM are quite basic, using a single core processor with 512 MB of RAM. In contrast, the RPi3 uses a quad-core processor with 1 GB of RAM. Future work will continue with the RPi3 for testing purposes. However, once a switch is made to the Pi Zero, the frame rate could drop down to 10-15 FPS. To solve this, it may be necessary to move to a more efficient language or devise a more efficient marker detection and tracking method that does not require an inordinate amount of processing power.

5.2 Marker Labeling

There remain errors in the method used for labeling markers. When labeling each marker, the program detects markers beginning with the bottom-left corner of the frame and moves to the top-right corner. Detection of each label is based on sequential position rather than true marker tracking. An array of markers is constantly being overwritten every frame. Part of the problem was the fact that markers were large. Future work will include designing a MoCap suit with smaller markers.

6 FUTURE TASKS

- Add a host computer to oversee the motion capture process.
- Install an on/of switch for each camera.
- Develop a "Boot and Go" system, where no direct, human interaction with the camera is needed.
- Install a battery pack for each camera.
- Develop a triangulation method for host computer.
- Integrate a gyroscope into each camera.

7 CONCLUSIONS

Initial work on this project has proven that using existing, low-cost hardware, and open source libraries, a motion capture system can be built for less than 1000 USD. Once this system is completed, all software and hardware designs will be disseminated open source via a suitable licensing scheme.

ACKNOWLEDGMENTS

Te authors would like to acknowledge the Hancock Tech/Art Fund at Western Connecticut State University for their support.

REFERENCES

2011. OpenCV image conversion from RGB to Grayscale using imread giving poor results. *Stack Overflow* (18 Sep 2011). Retrieved 07 Jun 2017 from https://stackoverfow.com/questions/7461075/opencvimage-conversion-from-rgb-to-grayscale-using-imread-giving-poor-results

2017. Face and Eye Detection Using OpenCv With Raspberry Pi. *Instructables* (30 Apr 2017). Retrieved 1 Jun 2017 from http://www.instructables.com/id/Face-and-Eye-Detection-Using-OpenCvWith-Raspberry/

Yuval Boger. 2013. What you should know about Head Trackers. *The VRguy’s Blog* (25 May 2013). Retrieved 22 Jun 2017 from http://vrguy.blogspot.com/2013/05/what-you-should-know-about-headtrackers.html

H.K.A. Devi. 2006. Thresholding: A Pixel-Level Image Processing Methodology Preprocessing Technique for an OCR System for the Brahmi Script. *Ancient Asia* (2006), 161âĂŞ165. Issue 1. http://doi.org/10.5334/aa.06113

R. Fisher, S. Perkins, A. Walker, and E. Wolfart. 2017. Thresholding. Image Processing Learning Resources. (12 Jun 2017). http://homepages.inf.ed.ac.uk/rbf/HIPR2/threshld.htm

Martin O’Hanlon. 2012. Raspberry Pi - run program at start-up. (10 Jun 2012). Retrieved 9 June 2017 from http://www.stufaboutcode.com/2012/06/raspberry-pi-run-program-atstart-up.html

Adrian Rosebrock. 2016. Install guide: Raspberry Pi 3 Raspbian Jessie OpenCV 3. *Pyimagesearch* (18 Apr 2016). Retrieved 29 May 2017 from http://www.pyimagesearch.com/2016/04/18/install-guide-raspberry-pi-3-raspbian-jessie-opencv-3/

Simon Sharwood. 2017. Raspberry Pi gives us all new ’Pi Zero W’ for its fifth birthday. *The Register* (28 Feb 2017). Retrieved 28 May 2017 from https://www.theregister.co.uk/2017/02/28/raspberry_pi_zero_w/