The Journal of Open Source Software

DynaMo: Dynamic Body Shape and Motion Capture with Intel RealSense Cameras

Abhishektha Boppana[1] **and Allison P. Anderson**[1]

1 Ann and H.J. Smead Department of Aerospace Engineering Sciences, University of Colorado Boulder **1**

DOI: 10.21105/joss.01466

Software

- Review
- Repository
- Archive

Submitted: 08 May 2019
Published: 29 September 2019

Background

Human body shape can be captured with a variety of methodologies, including laser lines, structured light, photogrammetry, and millimeter waves (Daanen & Haar, 2013). However, these technologies require expensive modules and have limited ability to capture dynamic changes in body shape.

Motion capture with specific markers is commonly done through camera-based motion tracking (Windolf, Götzen, & Morlock, 2008) These systems for marker tracking are often cost prohibitive and unable to capture surface morphology.

Recently, Intel released the D415 and D435 RealSense Depth Cameras, which use near-infrared structured light patterns and two infrared imagers to capture depth information at up to 90 frames per second. Purchasing a set of these cameras is more affordable than buying a dedicated motion-capture system for shape or marker tracking.

While Intel provides the `librealsense` library to interface with their cameras, it lacks tools to use multiple devices at once to capture shape and marker-tracking information. `DynaMo` builds upon `librealsense` to provide additional capability for researchers looking to capture such data.

`DynaMo` is designed to primarily assist those in the biomechanics and medical fields in capturing motion or body-shape data. It is currently being used in the Anderson Bioastronautics Research Group to capture dynamic changes in foot morphology.

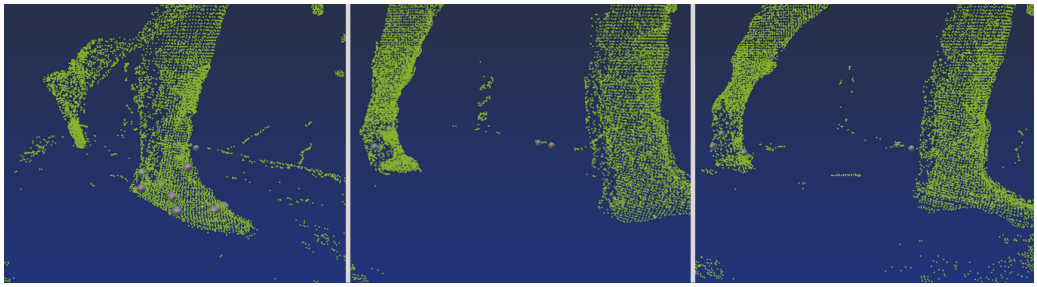

Figure 1: Sample frames collected by DynaMo showing dynamic shape capture (green) and marker identifaction (gray spheres)

Summary

`DynaMo` is a Python library that provides tools to capture dynamic changes in body shape and track locations of markers using Intel RealSense D4XX cameras. `DynaMo` was developed from

the examples provided by Intel in the Python `librealsense` library. It has been successfully tested streaming six cameras at 90 frames per second, all connected to one computer. `DynaMo` consists of several scripts that allow for calibration of multiple RealSense D4XX cameras to a common global coordinate system, simultaneous streaming of multiple RealSense D4XX cameras, viewing of data from multiple RealSense D4XX cameras in pointcloud format, and identification of reflecting markers from the pointclouds. The library is optimized to reduce the number of dropped frames while streaming.

`DynaMo` allows for the capture of depth, infrared, and color frames at an $(u \times v)$ resolution from Intel RealSense cameras. The values that are captured in each frame are listed below:

- Depth frames: s, where s is the distance to the object
- Infrared frames: Y, where Y is a single value from 0-255 denoting the monochrome pixel value
- Color frames: $[R, G, B]$, where R, G, B are red, green, and blue values, stacked to represent the color value of the pixel. This results in a $(u \times v \times 3)$ dimensional frame.

The pinhole camera model (Sturm, 2014) projects 3D points from the world $[x, y, z]$ onto a 2D image plane $[u, v]$ using the following formula:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \times \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

Where K is a matrix describing the camera's intrinsic properties, and s is the distance between the real-world point and the image plane. These properties include the focal length (f_x and f_y) and image offset (pp_x and pp_y) in each direction. They are represented in the matrix as:

$$K = \begin{bmatrix} f_x & 0 & pp_x \\ 0 & f_y & pp_y \\ 0 & 0 & 1 \end{bmatrix}$$

Since we are collecting 2D frames and we want to know the 3D location of the point to reconstruct the pointcloud, we can simply invert the K matrix and solve for the $[x, y, z]$ location as we know s, the distance between the 3D point and the 2D plane, and $[u, v]$, the coordinate of the point in the 2D plane:

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \frac{1}{f_x} & 0 & \frac{-pp_x}{f_x} \\ 0 & \frac{1}{f_y} & \frac{-pp_y}{f_y} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} s * u \\ s * v \\ s \end{bmatrix}$$

This transformation is known in the computer vision community, and is crucial to the functions present in `DynaMo`. `DynaMo` uses this transformation extensively in its calibration, streaming, and marker-tracking features.

Connected cameras are setup using a `device_manager` object which handles calls for communicating with the cameras. Cameras are first calibrated to a common global coordinate system by using a defined chessboard viewable by all cameras. The chessboard points are detected using the `findChessboardCorners` function of the OpenCV library (Bradski, 2000) for each camera's color image. Once the chessboard corners are found, they are translated to 3D points from the perspective of each camera and centered.

The Kabsch algorithm (Kabsch, 1976) is used to compute the (3×3) rotation matrix between each camera and the known chessboard coordinates. Translation is calculated by taking

the difference between the known chessboard corners and the camera's rotated chessboard perspective, resulting in a (3×1) matrix. The rotation matrix is horizontally stacked to the translation matrix, and a row of $[0, 0, 0, 1]$ is added to create a (4×4) matrix. This matrix transforms each camera's pointcloud from its local coordinate system to a global coordinate system.

Streaming is achieved by reading frames from each camera into a dictionary object saved in the computer's RAM. `DynaMo` checks frame numbers for continuity to ensure that frames are collected synchronously and are not repeated. Once streaming is complete, `DynaMo` aligns the images collected by the sensors in each camera to a common image center and saves the images as `pickle` objects to the disk. The data from all cameras can then be viewed as a single pointcloud for each frame from all cameras by using the previously computed transformation matrix.

A script is included to extract the locations of reflective markers on the pointcloud by simply thresholding for bright pixels on the infrared frame. Contours are then drawn fo each cluster of pixels on each camera's infrared frame; these contours highlight the detected markers by each camera. The center of each cluster is calculated and then translated into a 3D point using the depth frame. All points from all cameras are translated into a global coordinate system using the previously computed transformation matrix, and clusters are scanned for duplicates seen from multiple cameras.

Acknowledgments

This work was supported by a National Science Foundation Graduate Research under grant DGE 1650115. The authors would like to thank Dr. Rodger Kram and Dr. Wouter Hoogkamer for the use of their laboratory space for development and testing of the package.

References

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Daanen, H., & Haar, F. T. (2013). 3D whole body scanners revisited. *Displays*, *34*(4), 270–275. doi:10.1016/j.displa.2013.08.011

Kabsch, W. (1976). A solution for the best rotation to relate two sets of vectors. *Acta Crystallographica Section A*, *32*(5), 922–923. doi:10.1107/S0567739476001873

Sturm, P. (2014). Pinhole camera model. In K. Ikeuchi (Ed.), *Computer vision: A reference guide* (pp. 610–613). Boston, MA: Springer US. doi:10.1007/978-0-387-31439-6_472

Windolf, M., Götzen, N., & Morlock, M. (2008). Systematic accuracy and precision analysis of video motion capturing systems–exemplified on the vicon-460 system. *Journal of Biomechanics*, *41*(12), 2776–2780. doi:10.1016/j.jbiomech.2008.06.024