Universitetet
i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

MASTEROPPGAVE

Studieprogram/spesialisering: Automatisering og signalbehandling	Vårsemesteret, 2015 [✓ Åpen] / Konfidensiell
Forfatter: Yngve Finnestad	Yngve Finnestad (signatur forfatter)

Fagansvarlig: Morten Mossige

Veileder(e): Morten Mossige

Tittel på masteroppgaven:

Engelsk tittel: Analysis of Kinect Motion Capture Capabilities for Automated Robotic Painting Systems

Studiepoeng: 30

Emneord: Simplified Robot Programming ABB Kinect Motion Capture OpenCV	Sidetall: 53 + vedlegg/annet: source code Stavanger, 14. juni / 2015 dato/år

Analysis of Kinect Motion Capture Capabilities for Automated Robotic Painting Systems

Yngve Finnestad
202588

June 15, 2015

1 Abstract

The focus of this project is to determine the motion capture capabilities of the Kinect sensor. Motion capture is the process of recording human movement, and converting the captured data into a virtual environment. A simple implementation of motion capture is currently used by ABB for their Simplified Robot Programming project. Simplified Robot Programming allows the painter to program a robot by demonstrating the desired movement, instead of programming it.

To simplify the programming of paths for robotic paint applications, the movements of a professional painter are recorded and translated into a corresponding path in virtual space. The programmer will no longer have to manually program the coordinates in a path for the robot to follow, as it will follow the motion of the painter. This is currently done by a magnetic motion capture system.

This project will ideally determine if the Microsoft Kinect v2 sensor can serve as a more cost efficient replacement for the current system. The Kinect sensor features a 1080p camera and a time-of-flight based depth measuring technology. By combining these technologies, the goal of the implementation is to achieve tracking of a marker and compare the features of the Kinect to the current system.

The scope of this project extends to analyzing the accuracy and precision measured by the Kinect. This will only include the measured position of a marker, not its orientation, as this requires additional sensors or a different implementation.

To test the system's ability to track a dynamic target, the marker was mounted to an industrial ABB paint robot. The tracking and mapping capabilities of the Kinect worked to a satisfactory degree. While measuring a marker moving at 1 m/s the standard deviation of the measurement averaged at 2 cm, which should satisfy the requirements for a paint programming application.

Contents

2 Theory

The general purpose of a motion capture system is to record and convert movement data from a three-dimensional scene and, by image processing, convert it into relevant digital information. A motion capture system will generally consist of one or more markers located in the scene, and a sensor to capture their three-dimensional position, and in many cases their orientation.

This section will encompass the theory behind commonly used motion capture systems, and their intended use. Different implementations of motion capture systems will be presented and discussed, with a focus on advantages and disadvantages. Testing the Kinect device as a suitable motion capture sensor is an integral part of this project; hence its features and technology will be analyzed.

As image processing and computer vision is an important part of an optical motion capture system, the major steps in the process of capturing and analyzing data, using computer vision, will be discussed. The utilization of the collected data, and its relevance regarding the SRP project will be described near the end of this section.

2.1 Motion Capture

In the process of recording data from a scene and recreating it digitally, the first step is capturing multiple consecutive samples of the relevant movement in the scene. There are various systems designed for this purpose, which utilize a range of available technologies. As the primary purpose of this project is to measure the position of a single marker, a fairly basic motion capture system will suffice.

Multiple combinations of sensors and markers can be used for various motion capture purposes. Motion capture technology is widely used in special effects and entertainment applications, where the entire body of a person is tracked. The captured data is in these circumstances used to animate a virtual character in 3D.

This section will describe the general principle behind the most common motion capture systems.

2.1.1 Optical Systems

Optical motion capture systems are designed to triangulate the position of a marker covered by two or more image sensors. By setting up an array of two or more calibrated image sensors, the three-dimensional position of any marker in the covered area can be calculated.

The image sensors must be calibrated and must overlap the working area. The markers will then be captured by at least two of the image sensors. By segmenting the images to isolate the markers their position can be calculated with sub-millimeter accuracy. [1] (the process of triangulation and image processing will be discussed in 2.2.2 and 2.2.1 respectively.)

As motion capture technology has a wide area of use, the requirement of the system can change to better accommodate different scenarios. The two major variations in an optical motion capture system are in their use of passive or active markers. Both systems rely on a

[1] An Enhanced Correlation-Based Method for Stereo Correspondence with Sub-Pixel Accuracy http://perception.inrialpes.fr/people/evangelidis/george_files/ICCV_2005.pdf

light source, either infrared or visible light, but the placement of the light source vary between the two systems. Each type of light has its advantages and disadvantages. By equipping the markers and sensors to emit and capture IR light, there is less interference from visible light sources, which causes the markers to distinctively separate themselves from the rest of the scene.

The optical markers themselves contain no explicit information about their orientation, which gives each marker three degrees of freedom, as only the X, Y, Z axis can be directly observed. Orientation can, however, be inferred from the relative location of other markers.

Passive Markers

The markers used in a passive system will not contain any electronics, and will rely on their ability to be segmented when the image is processed. For the markers to separate themselves from the scene without being illuminated, they require a distinct feature to isolate themselves from the rest of the scene. For this purpose they will often be coated with a reflective material, or have a uniquely distinct color. When reflective markers are used in a controlled scene, they will be segmented by their high brightness, compared to the rest of the scene.

Reflective markers are widely used by motion capture actors, where markers are placed at strategic points to cover the movement of the whole body. This is to capture the real-time movements of the actor and translate it into a digital animation based on the actor's exact movements. When used in a professional setting, the recording will often take place in a motion capture studio, where the fixed scene around the motion capture actor can be controlled. As passive marker systems depend on being able to segment the markers from the rest of the scene, a specially designed background scene is optimal.

Passive markers do not have to be reflective. As long as a distinct feature of the marker can be isolated from the rest of the scene by segmentation, it can be used for motion capture. Naturally, to be able to easily isolate a marker based on a distinct feature, the background scene should not contain the same features. Tracking a blue ball in front of a blue sky will not be as effective as tracking the same ball in a field of grass, or in front of a white screen.

While passive markers offer a simple and cost-effective implementation of a motion capture system, it has its drawbacks. In a scene where the background can not be as easily controlled as in a studio, the markers might be harder to capture. As the emitted and reflected light follows the law of inverse squares, the light intensity of the markers can in some cases be insufficient. This is where the active marker becomes a better choice.

Active Markers

The active marker utilizes electronics to illuminate the markers by LED technology. As the markers are no longer reflecting external light, but provide their own, the useful range of the motion capture system is increased, as described by

$$Intensity = \frac{1}{distance^2}$$

A good feature of the active markers is their ability to emit a very specific kind of light, which can be tuned to make the segmentation more robust.

Every passive marker in the field of view of the camera will be illuminated at once, yielding multiple markers at once, which then has to be identified. A positive feature of active markers is their ability to be electronically controlled. Each individual marker can then be turned on or off at high frequencies. By illuminating one specific marker at a synchronized time, the motion capture system can distinguish the identity of each marker, but at the cost of a lower frame rate.

A further development of the active marker technology is referred to as *Time Modulated Active Marker Motion Capture*. For this method, each marker can be identified based on their pulse width modulated frequency.

Markerless motion capture

A newer technology in motion capture is markerless motion capture, which requires no markers but instead relies heavily on algorithms to classify human bodies. The Kinect device falls under this category, as it can identify up to six humans at once through an infrared and an RGB camera. After processing it will fairly accurately identify joint positions to generate a skeleton for each person it identifies. The Kinect will be further described in 2.3.

2.1.2 Non - Optical Systems

An alternative to optical systems is motion capture systems, utilizing magnetic or inertial measurement technology. These systems are not dependent on any visual markers but employ one or more electronic sensors.

Magnetic Systems

Magnetic motion capture systems emit a magnetic field from a source, where the markers will sense the relative change of the magnetic flux.

Magnetic motion capture systems have some advantages over optical systems. They do not require a line of sight, and can directly measure six degrees of freedom, giving both position and orientation. The drawback of magnetic motion capture systems is their negative response to metallic objects, which interferes with the sensors magnetic fields. The area of motion is also limited by the range of the magnetic field.

Inertial Measurement Systems

Motion capture system based on inertial measurement technology will use accelerometers and gyroscopes to record position and orientation respectively. Although the orientation is directly measured, the absolute position of the sensors is harder to estimate, as it will be derived from the acceleration. A common problem when estimating absolute position from acceleration only, is drifting. The sensor continuously integrates the current acceleration for all axes and angles, then adds the calculated velocity to the current velocity. The position is calculated the same way; by integrating the current velocity to estimate the current position. As the system continually adds changes in velocity and position, any eventual errors will also get added to the final position. If there is a consistent error in the acceleration measurement, this will propagate through the double integration and add as an error to the calculated position, which will accumulate as time goes. [2] , [3] A Kalman filter can be implemented to reduce noise error, but there will be a certain amount of drift in a system of this kind.

[2] Accelerometer for Mobile Robot Positioning
`http://biorobotics.ri.cmu.edu/papers/sbp_papers/integrated4/liu_accel_position.pdf`

[3] Drift-Free Position Estimation of Periodic or Quasi-Periodic Motion Using Inertial Sensors
`http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3231462/`

2.2 Computer Vision

For optical motion capture systems, the recorded data will consist of images of the scene. Computer vision and image processing is essential to process these images and separate the markers from the rest of the scene. The markers must first be separated from the rest of the scene by the process of segmentation, which will be discussed in 2.2.1. When the position of the markers has been determined in the captured images, their three-dimensional position can be estimated by the process of three-dimensional reconstruction from multiple images, which is discussed in 2.2.2.

2.2.1 Image Processing

To isolate the markers in the captured images from the rest of the scene, the image will go through a set of image processing techniques. Ideally, the scene of the motion caption recording will contain nothing that interferes with the features of the markers. Eg. if the tracked markers are blue, the scene should not contain any other blue objects if their blue color is the feature that separates them from the background. As there are many applications of motion capture, in various locations, the scene should not be expected to be ideal. To limit interfering factors in the scene and increase robustness, thresholding and smoothing are relevant techniques.

Image Segmentation

Image Segmentation is the process of dividing an image into multiple segments based on a defined feature. The purpose of the segmentation is to represent the parts of the image with common characteristics in an easy way. The image can be segmented with regards to features like brightness, color or texture. When applying this to motion capture, the object of interest will be the markers. As the purpose of the marker is to distinguish itself from the rest of the scene by exhibiting a distinct feature, the captured images can be segmented with regard to this exact feature. The background and markers will ideally represent different segments, where the markers can easily be distinguished from the rest of the scene.

There are multiple methods of image segmentation, and depending on the application and type of image, the method should be chosen accordingly. There exist two main categories of image segmentation, edge-based and region-based. Edge-based image segmentation classify regions based on a discontinuity between them, like a sharp change in intensity. Region-based segmentation classifies regions based on a common criteria, like the same intensity.

Thresholding

For the problem of isolating the marker from the background, the simple method of thresholding can be applied. Thresholding will segment a grayscale image based on pixel intensity. Each pixel in the image will be tested against a certain threshold value. If the pixel intensity falls above or below the threshold value, it will be classified as true or false accordingly, depending on the implementation. Thresholding an image will generate a new, binary image where the pixels classified as true will be white while the rest will be black.

While the principle of segmentation is simple, the hard part is to determine the correct threshold value. This value must be accurate enough to separate the object of interest, also called foreground, from the scene, or background. One approach is the use of a histogram to analyze the pixel intensity of the image, as illustrated in figure 1.

If there is a clear difference between the foreground and background, the histogram will show two distinct peaks, one for the foreground, and one for the background. The threshold value will then fall in between them. However, if there are no obvious distinction, the histogram will not show any clearly defined peaks, making the threshold value harder to pinpoint.

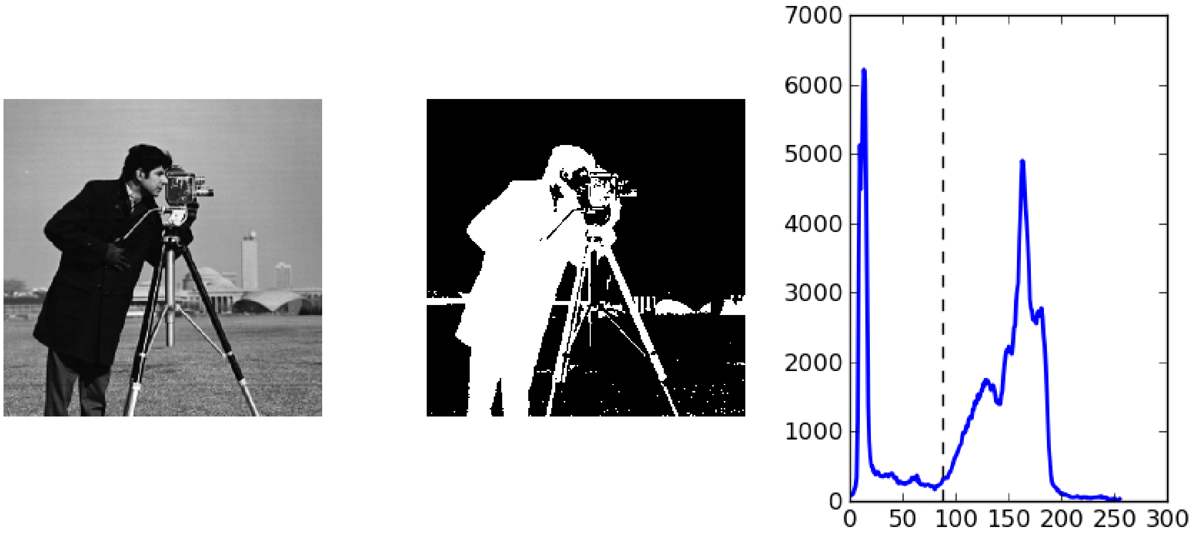


Figure 1: Illustration of a segmented grayscale image. The graph represents the quantity of the specific grayscale pixels, from 0 to 255, in this case, where the threshold value is set between the two peaks. [4]

Considering a normal motion capture scenario, the scene will ideally be designed to maximize the gap between the intensity of the marker (foreground), and the background. By using a marker with a characteristic easy distinguishable from the background, the segmentation of the image should be based on this characteristic. An example of this would be a reflective marker in a scene with a normal, uniformly lit background. The pixels representing the intensity of the marker would be higher compared to those representing the background. Looking at this in the form of a histogram, the markers would manifest as a peak in the high-intensity range of the histogram, although small, as the pixels representing the markers will be few compared to the background pixels.

However, if the targeted characteristic of the marker is a specific color, not intensity, segmentation by thresholding will be more complicated. As thresholding works by categorizing all pixels below a certain threshold on the intensity scale, as a segment, it is ideally suited for grayscale pictures. The pixels in grayscale pictures are single channel, meaning each pixel only has an intensity value between 0 and max resolution, which is 255 for commonly used 8bit pictures. To apply thresholding to color images, the thresholding must be done on specific channels of the image. Which channel to use depends on the color space used for the image.

[4] `https://scipy-lectures.github.io/packages/scikit-image/`

Color spaces

Color images can be represented in several ways, two common representations are RGB and HSV. RGB images display pixels as an addition of the primary colors; red, green and blue. Every color in the captured image will consist of a specific amount of the primary colors added together. RGB is a common representation, which is widely used. In image processing, each of these colors will have their designated *channel*. An RGB image will consist of three channels; red, green and blue (see figure 2).

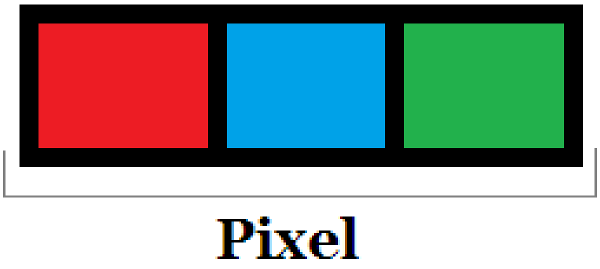


Figure 2: Illustration of the RBG components of a pixel. Each color represents one color channel. The combination of these three channel will represent one color when seen from afar. [5]

Describing a specific color range in the RGB color space is complicated, as each color in the spectrum is a result of three channels. As two shades of the same color can differ on all three color channels, mapping a range of a single color can prove difficult.

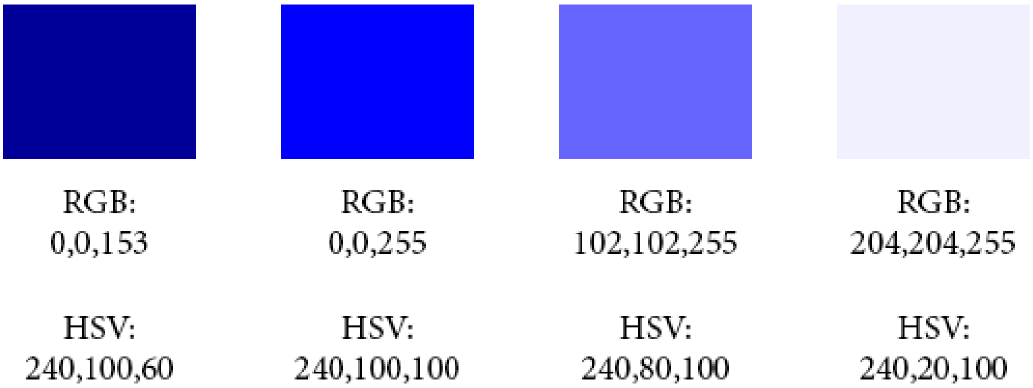


Figure 3: Illustration of the change in color channels between the RGB and HSV format.

The HSV color space is a derivative of RGB and makes defining a range of colors easier. HSV pixels are represented by a combination of hue, saturation, and value. By remapping the three-dimensional RGB, the HSV color space can be represented as a cylinder or cone. While the angle around the vertical center line corresponds to the hue, the saturation and value corresponds to the distance from center and distance in height respectively. To isolate a specific color and its different shades, a hue can be chosen with a narrow band while allowing larger bands on saturation and value.

[5] http://paginas.fe.up.pt/~ee06205/?paged=2

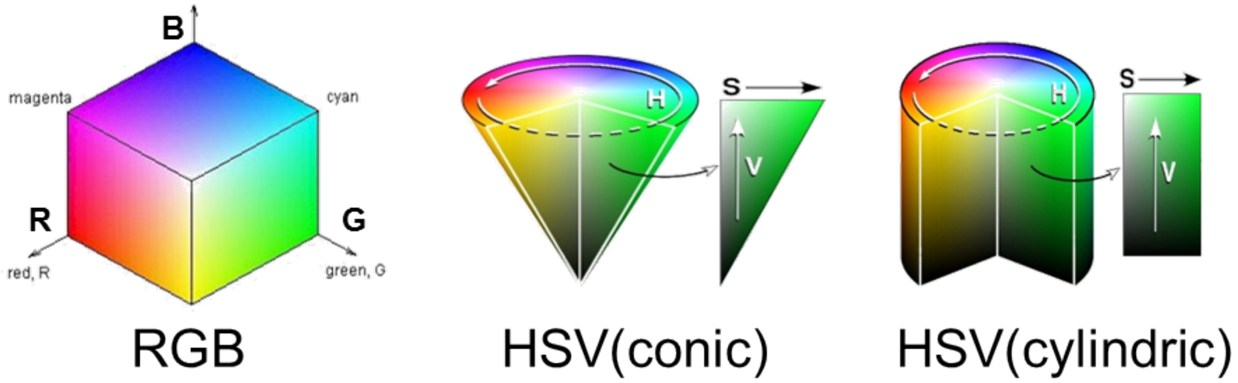


Figure 4: Representation of the RGB and HSV color spaces. [6]

This leads back to the threshold problem. As each image consist of multiple channels, the threshold must be applied for each channel of the image with respect to the characteristics of the channel. As the HSV color space separates the chromaticity from the saturation and brightness, a very specific chromaticity can be defined in the hue channel, while allowing a larger range of brightness and value.

In the case of tracking a marker of a specific color, the HSV color space gives the opportunity to easily define a hue that corresponds to the hue of the marker. As the marker may be recorded in varying degrees of lighting, a larger pass-band can be set for saturation and value.

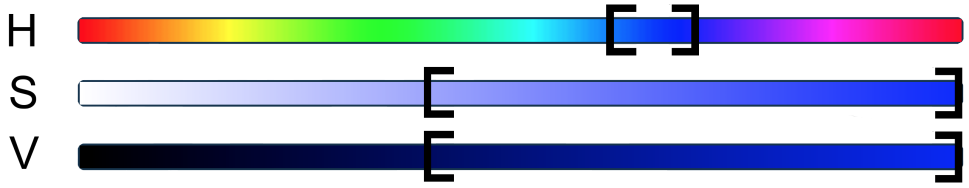


Figure 5: Representation of the passbands for the thresholding process. The brackets represent the upper and lower values, where hue has a narrower passband than saturation and value.

To isolate a specific color, the thresholding must be used as pass-bands for each of the channels. Everything above or below a range of intensities will be classified as irrelevant while the values falling in the pass-band range will classify as the marker.

Filtering

Before thresholding the image, it can be enhanced by a number of image processing methods. Even if the physical scene is optimized for motion capture, the captured images may contain noise and small details with characteristics similar to the markers. These errors can interfere with the segmentation process, where the noise may be classified as a false positive, resulting in a false marker detection. Image noise manifests as random variations in intensity and/or color in the image, and is usually caused by electronic noise in the image sensor.[7]

[6] `http://darkpgmr.tistory.com/66`

[7] Minimizing Electronic Noise in Digital Images
`http://conganat.uninet.edu/IVCVHAP/CONFERENCIAS/Alvira/index.html`

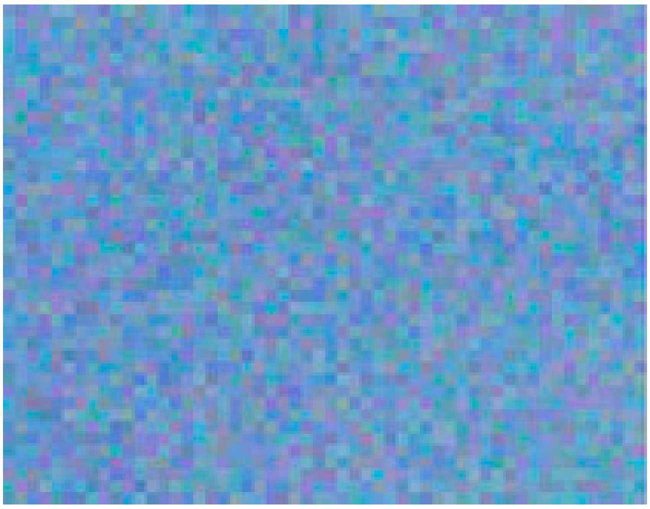


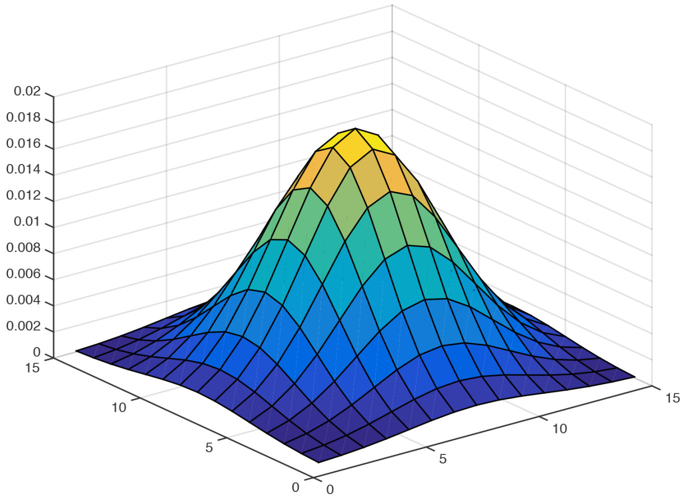
Figure 6: Example of noise in digital images. Note the variations of color manifesting through the noise. [8]

Image noise can coincidentally make pixels fall within the passband of the threshold, resulting in a false positive. However, as noise affects very small areas of the image, Gaussian smoothing can suppress much of it.

Gaussian smoothing is achieved through convolution of a Gaussian filter kernel and the image. The Gaussian filter kernel is a result of the Gaussian distribution function defined by equation 1. A good representation of the Gaussian model is given by the standard deviation function in one dimension, but in the case of image processing two dimensions are used as in equation 2.

$$G_{1D}(\sigma, x) = \frac{1}{2\pi\sigma} e^{-\left(\frac{x^2}{2\sigma^2}\right)} \qquad (1)$$

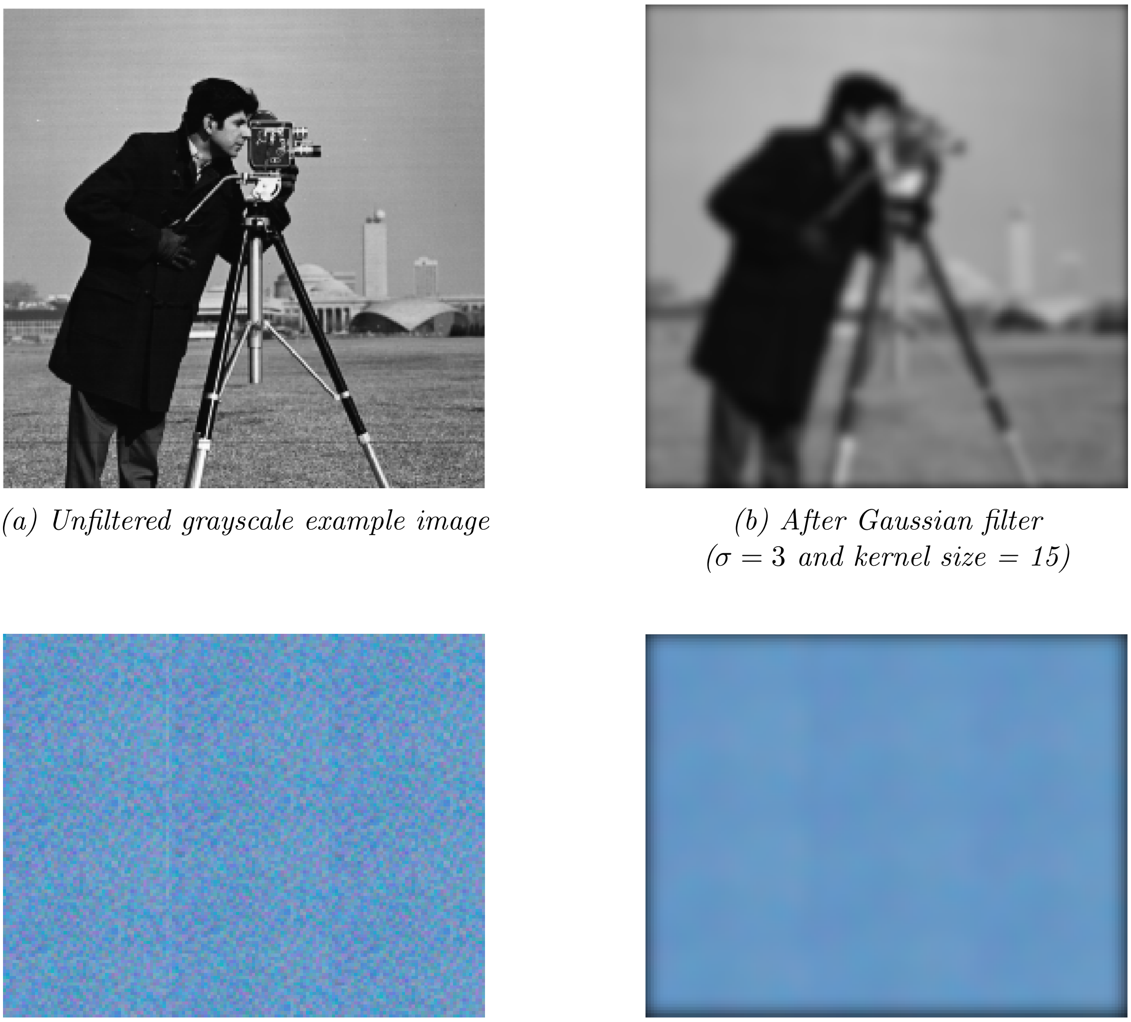
$$G_{2D}(\sigma, x, y) = \frac{1}{2\pi\sigma^2} e^{-\left(\frac{x^2+y^2}{2\sigma^2}\right)} \qquad (2)$$



Applying a Gaussian function as the filter kernel will result in a weighted average, where pixels closer to the center will have a higher effect on the result than pixels at a further distance, according to equation 2. As the image contains discrete pixels, the filter kernel will be a discretized approximation. The form of the kernel varies according to the standard deviation σ.

[8] `http://www.johnpaulcaponigro.com/blog/tag/noise/`

The standard deviation σ of the Gaussian function determines the extent of smoothing. A large standard deviation will utilize pixels in a larger region to compute the result compared to a smaller standard deviation. A very small standard deviation will yield very small smoothing effect compared to a larger standard deviation.



(a) Unfiltered grayscale example image

(b) After Gaussian filter ($\sigma = 3$ and kernel size = 15)

(c) Unfiltered high-noise color background.

(d) Filtered image, note the suppression of sharp colors pixels.

Figure 7: Illustration of the effects of Gaussian smoothing of images and noise.

Figure 7 illustrates the effect of Gaussian smoothing of an image. The small details, mainly noise, is suppressed leaving the large characteristics relatively intact. However, if the tracked object is detailed, a large standard deviation may smooth the wanted details of the object as well as the disturbances. The size of the mask should be large enough to contain the entire Gaussian function, as a too small kernel size will disregard the outer edges of the Gaussian function.

Image Moments

After thresholding the captured images, the only remaining segments will ideally be the markers. The next step in the image processing chain is locating the center of the segments. An approach to finding the segment center is the use of image moments.

Image moments are a form of weighted average, based on the intensity of the pixels in the image. The function for determining image moments are described by equation 4 and 5, which represent the continuous and discrete versions respectively. The general function for image moments from a point $[c_x, c_y]$ is described in 3, where m, n represents the orders.

$$\mu_{m,n} = \iint (x - c_x)^m \, (y - c_y)^n \, f(x,y) \, dy \, dx \tag{3}$$

As the moments are calculated from $(0,0)$, c_x and c_y will be zero. This gives the continuous (4), and discrete image moments (5) functions. For the discrete function (5), the summation will span the height h and width w of the image.

$$\mu_{m,n} = \iint x^m \, y^n \, f(x,y) \, dy \, dx \tag{4}$$

$$\mu_{m,n} = \sum_{x=0}^{w} \sum_{y=0}^{h} x^m \, y^n \, f(x,y) \tag{5}$$

Assuming the thresholded image is binary, the zeroth order moments will simply add the true pixels in each dimension, and will calculate the area of the segmented region. The center of the segment can be found by also calculating the first order moments for each dimension. The first order moments will accumulate the coordinates of the true pixels, according to equation 5.

The spatial center of the region, or the *center of gravity* can be described by equation 6, where the first order moments are divided by the area found by the zeroth order moments.[9]

$$\begin{aligned} x_{center} &= \frac{\mu_{1,0}}{A} = \frac{\mu_{1,0}}{\mu_{0,0}} \\ y_{center} &= \frac{\mu_{0,1}}{A} = \frac{\mu_{0,1}}{\mu_{0,0}} \end{aligned} \tag{6}$$

Image moments must be applied to each isolated segment individually, as it will only be effective for one segment in an otherwise empty subsection of the image. If there are more than one segment, the center of gravity will fall between the two. This is why noise suppression is a valuable step in the process, as unsuppressed noise can manifest as separate segments, which will affect the center moments.

[9] Simple Image Analysis By Moments
`http://breckon.eu/toby/teaching/dip/opencv/SimpleImageAnalysisbyMoments.pdf`

2.2.2 Aquiring Depth by Stereo Vision

Image processing is one aspect of motion capture, and will ideally find the center of the markers in the image. However, to relate this two-dimensional position within the image to the real world scene, and acquire the depth of the marker, stereo vision can be applied. With minimum two cameras covering the same area of the scene, the position of a common detected object can be acquired relative to one camera.

There are two main coordinate spaces to consider. The first is the internal coordinate space of each camera, where the *(x,y)* coordinates are known, as they were calculated during the image processing phase. The second coordinate space is the *world space*, which is the actual coordinate space of the scene, and is the relevant space to measure movement. To simplify the explanation, a scene with one marker and two cameras can be considered as in figure 8.

The marker will be located at a three-dimensional point in the scene *P(X,Y,Z)*, where *(X,Y,Z)* are world-space coordinates. The purpose of stereo vision is to calculate the position of the marker based on the gathered info from the two cameras. The marker position in the camera space can be described as *u(x,y)*. The camera setup for stereo vision systems will often consist of two cameras placed alongside each other, giving the same y coordinate. For a setup similar to the one in figure 8, where the orientation of the cameras will be identical, the position of the marker in camera space will be expected to differ along the x-axis, as there is a distance in x-axis between the cameras.

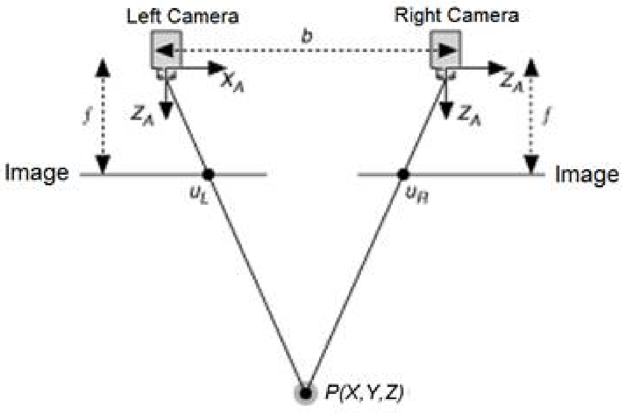


Figure 8: Simple stereo camera setup. [10]

The same point is viewed by both cameras, which is separated by a distance b on the X axis, the horizontal points of the marker in camera space can be described as u_L and u_R for the left and right camera respectively. The camera focus f is an intrinsic constant of the camera, and is camera dependent.

$$p_L = f \cdot \frac{X}{Z} \qquad p_R = f \cdot \frac{X-b}{Z} \tag{7}$$

[10] http://www.ni.com/white-paper/14103/en/

From the distance between the points u_L and u_R, in its respective images, the disparity d can be calculated. The disparity is the camera space distance between the markers located in the images.

$$d = u_L - u_R = f \cdot \frac{b}{Z} \tag{8}$$

solving for Z, which is the world-space depth gives.

$$Z = f \cdot \frac{b}{d} \tag{9}$$

This is the fundamental principles behind stereo vision. It demonstrates a very simple setup and essential theoretical steps. Any applied stereo vision setup will be far more complex and must include a camera calibration process. The essence of calibrating a camera is determining the camera matrix, which describes the mapping from a 3D point in world-space to a corresponding 2D point in the captured images.

$$\textit{image plane coordinates}: \quad p = \begin{bmatrix} x \\ y \end{bmatrix} \qquad \textit{world coordinates}: \quad {}^cP = \begin{bmatrix} {}^cX \\ {}^cY \\ {}^cZ \end{bmatrix} \tag{10}$$

$$\begin{aligned} \textit{optical axis}: &\quad {}^cZ \\ \textit{camera axis}: &\quad {}^cZ \\ \textit{camera center}: &\quad {}^cX =^c Y =^c Z = 0 \end{aligned}$$

From (10) the coordinates can be written as, where f is the focal length of the camera:

$$\underbrace{\begin{bmatrix} x \\ y \\ f \end{bmatrix}}_{\text{p}} = \frac{f}{{}^cZ} \underbrace{\begin{bmatrix} 1\,0\,0\,0 \\ 0\,1\,0\,0 \\ 0\,0\,1\,0 \end{bmatrix}}_{\Pi_0} \underbrace{\begin{bmatrix} {}^cX \\ {}^cY \\ {}^cZ \end{bmatrix}}_{{}^cP} \tag{11}$$

Introduce normalized image coordinates by dividing by *f*.

$$\hat{x} = \frac{x}{f} \qquad \hat{y} = \frac{y}{f} \tag{12}$$

This gives (13) where x and y are measured in metric distance.

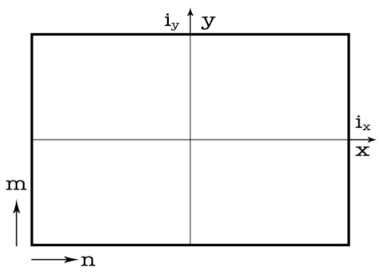
$$\hat{p} = \begin{bmatrix} \hat{x} \\ \hat{y} \\ 1 \end{bmatrix} = \frac{1}{{}^cZ} \Pi_0 \, {}^cP \qquad \begin{aligned} x &= f\hat{x} \\ y &= f\hat{y} \end{aligned} \tag{13}$$

The metric size of the pixels is Δx, Δy, where the image coordinates in pixels will then be

$$\begin{aligned} i_x &= \frac{x}{\Delta x} = \frac{f}{\Delta x} \cdot \hat{x} = \alpha \cdot \hat{x} \quad \text{when} \quad \alpha = \frac{f}{\Delta x} \\ i_y &= \frac{y}{\Delta y} = \frac{f}{\Delta y} \cdot \hat{y} = \beta \cdot \hat{y} \quad \text{when} \quad \beta = \frac{f}{\Delta y} \end{aligned} \tag{14}$$

This reduces the intrinsic parameters $f, \Delta x, \Delta y$ to the constants α and β. By using the lower, left corner of the image as a starting position for the image indexes (m, n)

$$\begin{aligned} n &= i_x + n_0 \\ m &= i_y + m_0 \end{aligned}$$



this can then be described as

$$\underbrace{\begin{bmatrix} n \\ m \\ 1 \end{bmatrix}}_{\text{p}} = \underbrace{\begin{bmatrix} \alpha & 0 & n_0 \\ 0 & \beta & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{K} \underbrace{\begin{bmatrix} \hat{x} \\ \hat{y} \\ 1 \end{bmatrix}}_{\hat{p}} \qquad p = K\hat{p} \tag{15}$$

Combining this gives a function for p,

$$p = K \, \frac{1}{{}^cZ} \, \Pi_0 \, {}^cP \qquad \rightarrow \qquad {}^cZ \, p = K \, \Pi_0 \, {}^cP \tag{16}$$

which can be solved for ${}^cZ$, giving the metric distance from the camera to the object. Further, let

$${}^cZ = \lambda \quad \text{and} \quad {}^cP = TR \, {}^wP \tag{17}$$

where ${}^wP$ represent the world coordinates, as reference to the real world scene.

$$TR = T \cdot R = \begin{bmatrix} R & t \\ o^T & 1 \end{bmatrix} \qquad t = \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix} \tag{18}$$

The TR matrix is a Translation-Rotation matrix, which describes the camera center in relation to the "real-world center", which is defined as the origin point for the scene. R is a conventional three-dimensional rotation matrix and describes the orientation of the camera, related to the scene coordinate system. T is a translation vector, and describes the linear translation in each axis from the scene center. This gives

$$\lambda p = K\Pi_0\, TR\, {}^{w}P = M\, {}^{w}P \qquad \textit{where} \qquad M = K\Pi_0\, TR \tag{19}$$

which can be written as

$$\Pi_0\, TR = \begin{bmatrix} I_d & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} = \begin{bmatrix} R & t \end{bmatrix} \tag{20}$$

The camera matrix is then

$$M = K\ \begin{bmatrix} R & t \end{bmatrix} \tag{21}$$

K contains all intrinsic parameters, while $\begin{bmatrix} R & t \end{bmatrix}$ contains all extrinsic parameters.

As displayed, the intrinsic and extrinsic parameters must be supplied, for the calculation of the camera matrix to be calculated. The intrinsic parameters needed are focal length f, pixel width Δx and pixel height Δy. The intrinsic parameters are camera specific and have no relation to the scene.

The extrinsic parameters are directly related to how the camera setup is configured, where the rotation and translation matrix relate the camera orientation and position to the defined world coordinate system respectively.

There are multiple methods of acquiring these parameters. Special software and Matlab packages are designed for this exact purpose. A common technique is to use a reference template, which will be captured by both cameras in the scene. As the reference template contain figures with known dimensions, the software will be able to estimate the parameters needed for camera calibration.

2.3 The Kinect Device

The Kinect device is a motion sensing device developed by Microsoft primarily for the Xbox game console. Kinect version one was announced in 2010 while version two was released in 2014. The purpose of the Kinect sensor is to enable a person to interact with their Xbox through physical movement. Contrary to previous systems created for the same purpose, the Kinect has a depth measurement feature. The depth measurement sensor is an addition to the RGB camera, and allow the Kinect to function as a standard camera, but also as a depth measurement system.

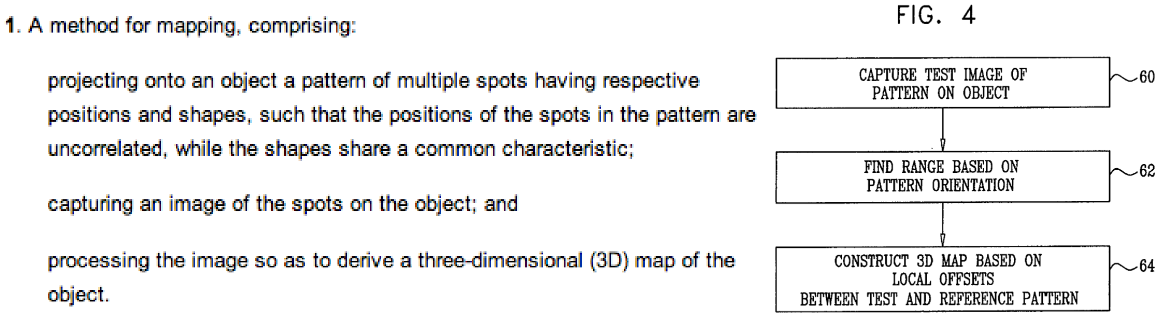
Figure 9: The Microsoft Kinect v2 [11]

The optical aspect of the Kinect device contains an RGB camera, IR projection device, and an IR camera. The IR camera work in conjunction with the IR projection to estimate the depth of the scene in view. The RGB camera has no direct relation to the IR system or the depth measurement system, and is simply used as a conventional camera.

The principle of acquiring depth differs between the two versions of the Kinect. Version one uses a sensor developed by Primesense, which produces a 320x240 pixel grayscale depth map. An infrared pattern will be projected on the scene, which will then be captured by the infrared camera, then processed using the proprietary LightCoding technology. All depth computation is done on embedded, dedicated hardware in the Kinect device. This will provide an 11-bit depth map, resulting in a depth resolution of 2048.

The exact details of the depth calculation are hard to pinpoint, as it runs on proprietary software, but the key points of the depth mapping process can be shown from the US Patent for this method (figure 10).

[11] `http://commons.wikimedia.org/wiki/File:Xbox-One-Kinect.jpg`



1. A method for mapping, comprising:

projecting onto an object a pattern of multiple spots having respective positions and shapes, such that the positions of the spots in the pattern are uncorrelated, while the shapes share a common characteristic;

capturing an image of the spots on the object; and

processing the image so as to derive a three-dimensional (3D) map of the object.

FIG. 4
CAPTURE TEST IMAGE OF PATTERN ON OBJECT
60
FIND RANGE BASED ON PATTERN ORIENTATION
62
CONSTRUCT 3D MAP BASED ON LOCAL OFFSETS BETWEEN TEST AND REFERENCE PATTERN
64

(a) Brief explanation of the structured light technology used by Kinect v1

(b) Diagram of depth mapping process for the structured light technology.

Figure 10: Excerpts from the Kinect v1 depth mapping patent (US 20080106746 A1), showing the major steps in depth measuring by structured light. [12]

As shown in the patent, the first version of the Kinect utilizes structured light imaging. The IR emitter projects a pseudo-random grid of dots, illuminating the field of view. The image processing hardware on the Kinect will analyze this field of dots, and establish a relationship between them. Any movement in the scene will distort this relationship, and provide enough data to determine the corresponding depth for that area. As the depth calculation depend on the relationship between these dots, it puts a limit in resolution. Large objects will be covered by several dots, and thereby give a better representation of its shape compared to a smaller object, where there are an insufficient amount of dots covering the object.

This is generally not a problem for the intended use of recording the motion of humans for the purpose of entertainment, as small details are insignificant in this application. However, this can be a bigger problem when picking up smaller objects, such as a motion capture marker.

The Kinect version 2 are fairly similar to its previous version. The purpose and general principle behind the Kinect as a gaming device remain the same, but there are significant hardware upgrades. The new sensor features a 512x424 pixel depth map and a 1080p camera. It will process up to two gigabits of data per second, and interfaces with Windows 8 machines and Xbox One consoles through USB 3.

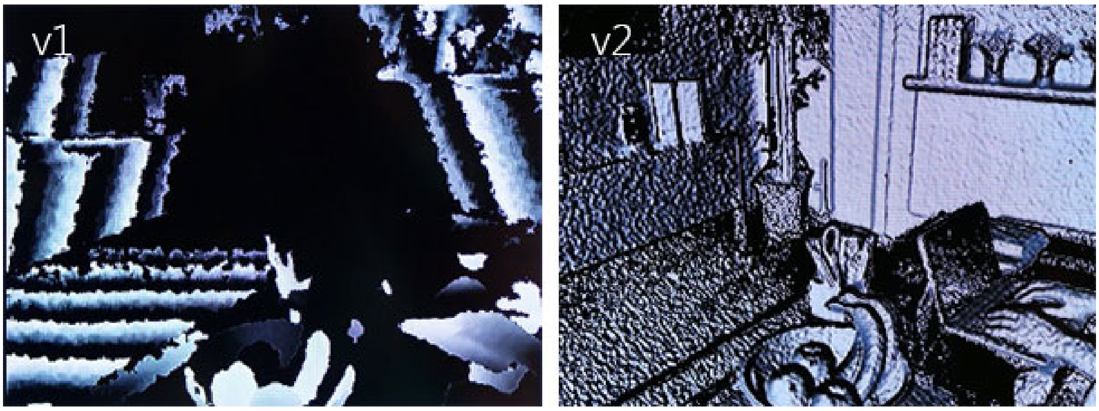


Figure 11: Demonstration of the depth measurement quality of the Kinect v1 (left), where structured light is used, and the Kinect v2 (right), where time-of-flight is used. [13]

[12] http://www.google.com.ar/patents/US20080106746

[13] https://www.microsoft.com/en-us/kinectforwindows/meetkinect/features.aspx

The depth measurement method for version 2 is based on *Time-of-Flight* technology. Time of Flight technology relies on projected light and the time it takes to reflect from a surface. The light emitter will emit a pulse of light which will reflect from the surface of the scene, and back to the sensor. An array of sensors will capture the reflected light, but as there are differences in the depth of the scene, there will be a delay for some of the returning light. This delay can be described as

$$t_D = \frac{2D}{c} \tag{22}$$

This outlines the main difference between the two technologies, where the earlier version relies on the change in a neighborhood of dots, the newer version captures the depth for each pixel, increasing the resolution.

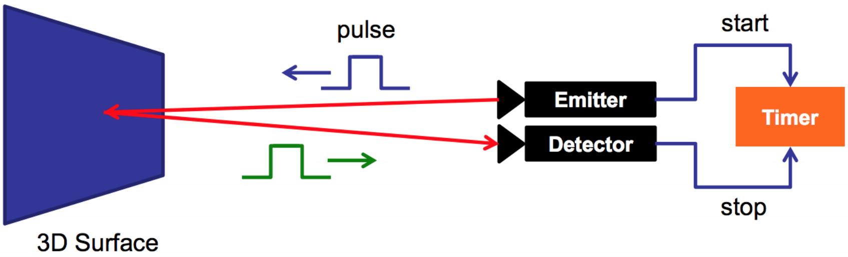


Figure 12: Illustration of the principle behind time-of-flight technology [14]

There are some disadvantages regarding this type of system. Unwanted reflection and interference are two significant ones. Interference occurs when multiple devices observe the same space. As all devices rely on the same measurement method, interference will disrupt the timing process. Multiplexing or modulation can, however, fix this. The other significant disadvantage is reflection from the scene. As the light can be reflected from a different point than the one intended, or reflected away from the measured point, is can cause local errors in depth measurement.

The major advantages of time-of-flight systems include the size, simplicity, and efficiency. Compared to a stereo vision system, where each captured image needs to be processed to find disparity, and then distance, a time-of-flight system will retrieve all depth data with a single scan. As a camera rig is no longer required, the size of the system can be considerably compressed to the point where everything needed can be embedded in a single unit, like the Kinect. A stereo vision rig will also require extensive calibration, where a small error can severely affect the end results. A time-of-flight device requires no similar calibration, except for its extrinsic parameters, when related to a reference point in the scene.

2.4 3D Path Creation

After the 3D position of the marker has been acquired, a path will be generated. In ideal conditions, the Kinect will capture images at a rate of 30 frames per second. Combining the raw captured data without further processing might give an unsatisfactory path, as the data might be affected by noise from either the sensor itself or the image processing. Occlusion of the marker may give time segments without any positional data, which must be taken into consideration.

[14] http://campar.in.tum.de/twiki/pub/Chair/TeachingSs11Kinect/2011-DSensors_LabCourse_Kinect.pdf

There are a number of suitable methods for improving this kind of signals. Moving Average is a common signal processing method for removing disturbances in a signal. Moving Average is a low pass filter that suppresses sudden, rapid changes in the positional values recorded, as such rapid deviations often are manifestations of noise. Below is a formula for the moving average, with output signal y and input signal x. M is the window size of the moving-average filter.

$$y[i] = \frac{1}{M} \sum_{j=0}^{M-1} x[i+j] \tag{23}$$

A time stamp can be put on each acquired point, which gives an estimate of the velocity between two samples. This can serve as a data set of positional data for the Simplified Robot Programming algorithms, which will generate a path for the robot to follow.

2.5 Simplified Robot Programming

The Simplified Robot Programming (SRP) technology aims to cut programming time, and complexity when creating robotic paint programs. The goal is to replicate the motions of a human painter and translate it to a robotics paint program. This process starts with recording the motion of the painter, where a handheld sensor, similar to a conventional spray-paint pistol is used. The captured data will contain position, orientation, and time stamps. This allows for the recreation of the near exact motion of the painter, but in virtual space.

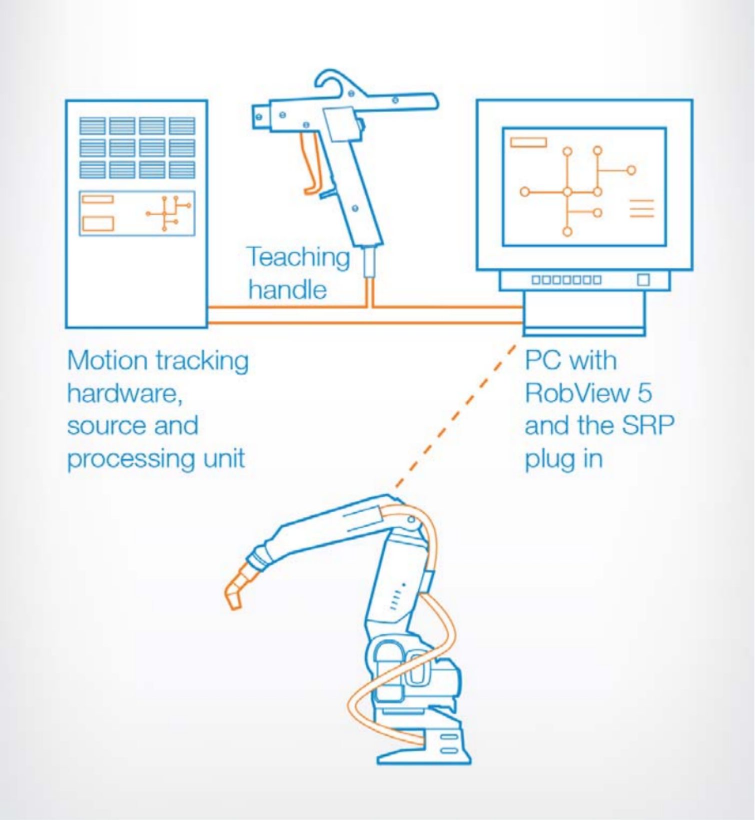


Figure 13: Illustration of the SRP setup [15]

[15] https://library.e.abb.com/public/8e8168587cb1ef4cc1257ddc0033de9f/SRP_Final_external.pdf?filename=SRP_Final_external.pdf

Algorithms will process the captured data, and optimize it for use in a robotics paint program. Once the processing is done, the robotic manipulator will move its spray-paint tool in a path consistent with the one recorded.

The SRP project will both simplify and advance robotic paint programming, as it removes the need for manually programming a path, with can be both complex and time consuming. The advancement comes as a result of capturing the authentic human movement, instead of computer programmed coordinates. The human aspect of painting will be transferred to the final path, giving it a more complex and fluid movement.

2.5.1 The SRP Process

The process starts by recording the painters movement. This is done by the Polhemus Liberty system, a magnetic based motion capture system. The Polhemus system itself will later be described in section 2.5.2, while the general principle of magnetic motion capture systems is discussed in 2.1.2. The painter will move a hand-held sensor, resembling a standard spray-paint pistol with a magnetic sensor embedded. Buttons on the pistol enable the painter to start and stop the sequence while the system continuously record the position and orientation.

After capturing the data, algorithms will simplify the captured points, down to what is needed for the robot to follow the path to a satisfactory degree. The motion capture system will capture a huge amount of samples, where all may not be as relevant. The information they provide might be superfluous to the intended path.

An example of this is defining a straight line by more than two points. As only two points are needed to form a line segment, any excess points are irrelevant to the form of the line itself. However, the sampled points contain a timestamp, which will describe the velocity between points. Removing points will then simplify the path, but may also cause loss of useful velocity data. This is a trade-off between a simple and easy robot path with a few points, and a more detailed path with more points to take into consideration. This must be taken into consideration when forming the robot path, as simple paths with few points are easier to manipulate if needed. The path can be viewed and edited in *RobView*, a software developed by ABB for the purpose of viewing and tuning robots in a painting setting. Finally, the path will be transferred to the paint robot controller, as a paint program for the robot to follow.

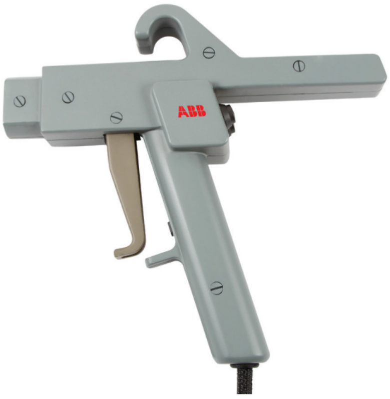


Figure 14: The tool marker currently used in SRP. [16]

[16] https://library.e.abb.com/public/ba0082be61601cb9c1257ddc003ad796/Simplified%20Robot%20Programming_data%20sheet.pdf?filename=Simplified%20Robot%20Programming_data%20sheet.pdf

2.5.2 The Polhemus System

The Polhemus system utilizes magnetic motion capture to track the position and orientation of a marker, as a general magnetic motion capture system do. The system uses a proprietary *AC electromagnetic technology*, and will deliver points with six degrees-of-freedom, as it will calculate position (X, Y, Z) and orientation (Pitch, Yaw, Roll). The system has a high sample rate compared to conventional optical motion capture systems.

The major advantage of the Polhemus system, and magnetic motion capture systems in general, is their ability to track the marker regardless of occlusion of the marker, which is a major disadvantage in optical motion capture systems. However, like all systems based on magnetic fields, the Polhemus system is vulnerable to metallic objects, as these will distort the magnetic field emitted by the source. As the magnetic field is distorted, the readings of the sensor will be affected accordingly.

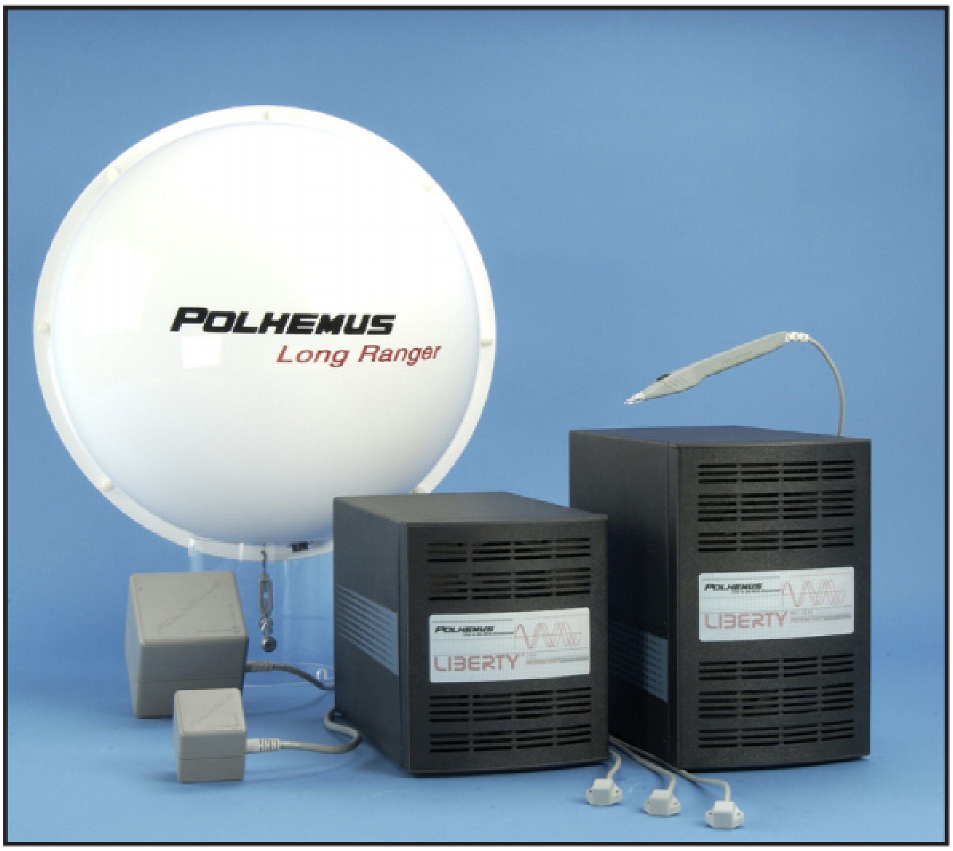


Figure 15: The Polhemus Liberty product line, used in the SRP project [17]

[17] http://polhemus.com/_assets/img/LIBERTY_Brochure.pdf

3 Implementation

This section will describe how a simple motion capture system was implemented. The purpose of implementing this system is to analyze its motion capture capabilities. The final implementation will be a combination of the different methods and techniques described in the theory section. It will be able to capture the 3D position of a marker by using only the Microsoft Kinect v2 (further referenced as Kinect) as a motion capture sensor. A simple demo video has been made for the purpose of illustrating the process, and can be viewed at `https://youtu.be/yjUPy5moT5I`

3.1 System Setup

The purpose of a motion capture system is to track the 3D position of markers in a scene. In this case, the system will only track one marker. This marker can be mounted to a spray-paint pistol sensor, as it is in the SRP project described in 2.5.

The purpose of this implementation is to acquire the positional data of the marker. Orientation is outside the scope of this project, as it is not achievable with this setup of only one marker and no inertial measurement systems. The marker will be an illuminated blue ping-pong ball, as this inhibits good features for a marker, as described in 3.3.1.

The Kinect is the focus of this project, and will be the only device used in this implementation. As the Kinect contains both a color camera and depth measurement technology, these two technologies will be combined to form the final result. The RGB camera (further referenced as the color camera) will capture a color image of the scene and process the image to isolate the position of the marker within the image. This is the same method used in stereo vision systems, as described in 2.2.1, but after determining the position of the marker in only one image, it will be used for a different purpose than determining disparity, as is the next step in stereo vision systems.

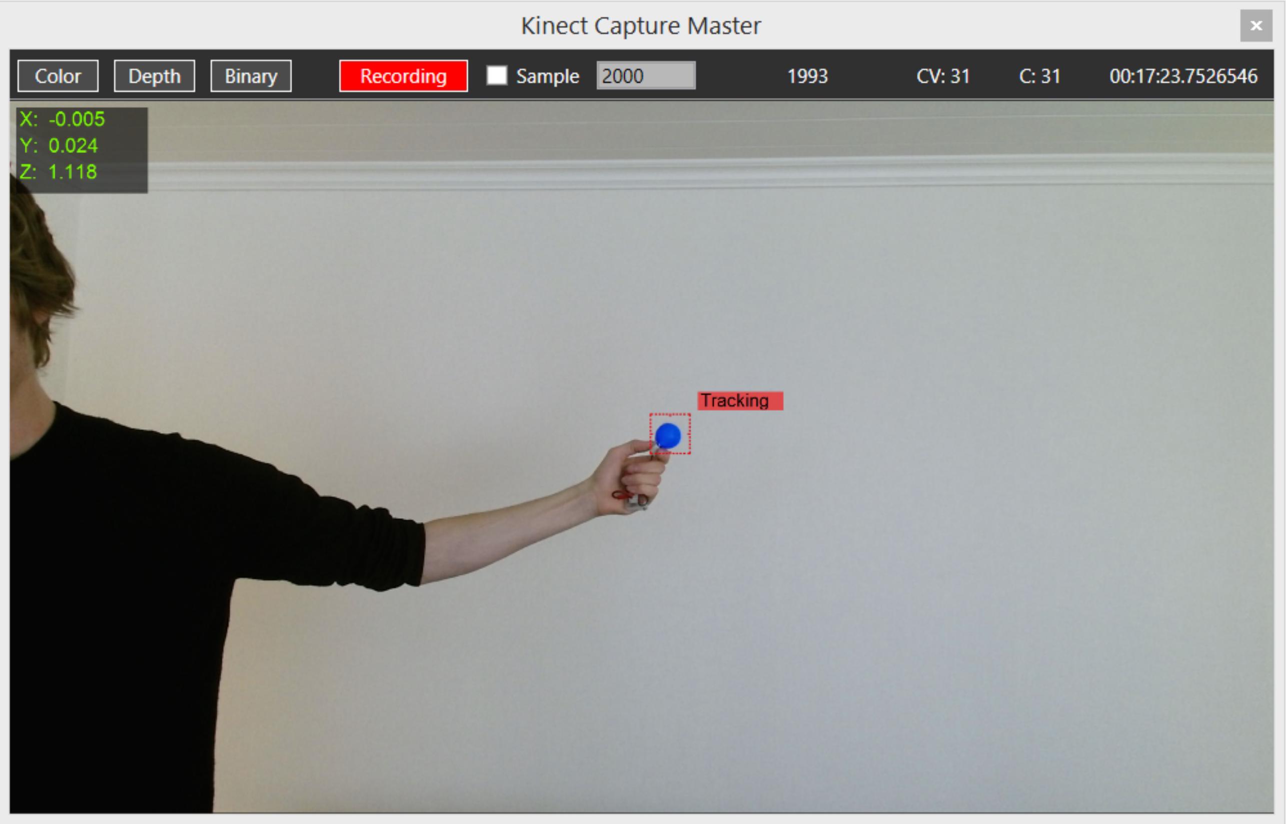
As the Kinect will additionally capture a grayscale depth image (further referenced to as depth image) of the scene the 2D coordinates from the color image will be mapped to its corresponding location in the depth image. By reading the depth value at this position, the depth can be determined as well.

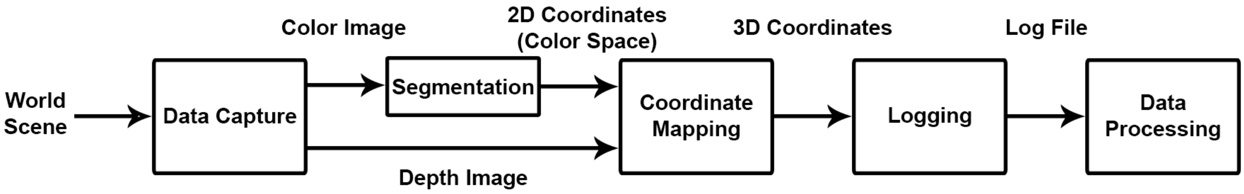
A 3D position has now been found, and will be logged in a log file. This file can then be processed. In this implementation, the data will be processed as described in 3.2.5. This entire process will be described in further detail, as the software implementation will be presented next.

3.2 Software Implementation

The majority of the software implementation is done in *C#*. This allows for easy use of the Kinect SDK, created by Microsoft for the purpose of developing applications for the Kinect. The image processing is done by OpenCV, a powerful image processing library often used for real-time applications. As this application is written in *C#*, while OpenCV is originally written in C++, EmguCV, a wrapper for *C#* is used in this implementation. After the *C#* application has logged the position, it will be written to a text file that will be processed by a Python script for the purpose of signal processing, and graphical representation.

Below is a screen shot of the application, where the view of the color camera is shown by default. An indication will be displayed over the currently tracked marker, where the user can change the tracked color by clicking anywhere on the image, and thereby set this color as the new tracked color. There is a start / stop record button, which will start and stop the current recording session.



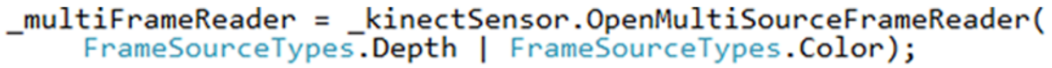


The figure above shows the major steps in the process. Each of these steps will be discussed next.

3.2.1 Data Capture

The data capturing is event triggered. Using the Kinect SDK Library, subscriptions to certain events can be configured.

The application subscribes to the *MultiSource* event, which triggers when Kinect has captured and processed all specified types of input. These inputs include images captured from the variety of sensor on the Kinect, such as color, infrared- and depth images. The Kinect can also supply sound and an estimation of certain joint positions of the person in the scene. The functions used in this implementation are the color- and depth image. By configuring this event, the Kinect will supply a *depth frame* and a *color frame*, which are objects storing the depth and color data respectively.

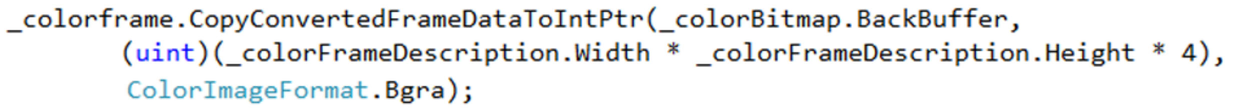


```
_multiFrameReader = _kinectSensor.OpenMultiSourceFrameReader(
    FrameSourceTypes.Depth | FrameSourceTypes.Color);
```

These objects contain more data than just the raw image data. They provide valuable information about the settings of the sensor as the specific image was taken, as well as details regarding the image itself.

In optimal conditions, the Kinect will capture both depth and color at 30 frames per second. In low-light conditions, the color camera will drop to 15 frames per second. As the subscription is set up to receive both depth and color, as both are needed, this will limit the total frame rate, even though the depth sensor works at 30 fps.

As an event triggers, the raw data extracted from the objects returned from the Kinect will be copied to specific memory locations, referenced to by pointers. The raw data of the color frame will be stored in the *backbuffer* of a *writable bitmap*. The Writable bitmap is part of the *Windows.Media.Imaging* library, and provides the opportunity to store specified pixel data in a back buffer, which keeps a consistent memory address. Storing the image data in a consistent memory allocation and rewriting it, is more efficient, compared to copying the image to a new memory location.



```
_colorframe.CopyConvertedFrameDataToIntPtr(_colorBitmap.BackBuffer,
        (uint)(_colorFrameDescription.Width * _colorFrameDescription.Height * 4),
        ColorImageFormat.Bgra);
```

The *CopyConvertedFrameDataToIntPtr(...)* method will copy the raw pixel data of the color image to the memory allocation indicated by the pointer. The first argument is the data to copy. As mentioned, the back buffer, which only contains the raw data is copied. The next argument describes the size of the memory allocation needed. As this is a BGRA image (Blue Green Red Alpha), it will require $Width \cdot height \cdot 4$ bytes to contain all four image channels. The last argument simply describes the current image format.

While this method in itself seems insignificant, it illustrates the images format and the storage method used in this implementation. When discussing the image processing in 2.2.1, the importance of pointers will be further discussed.

Storage of depth data follows the same procedure. The raw depth data is copied from the *underlying buffer* of the depth object. This represents the raw depth data of the depth image. Raw data is used for visualization purposes in the GUI, but will also be used for coordinate mapping purposes, as described in 3.2.3.

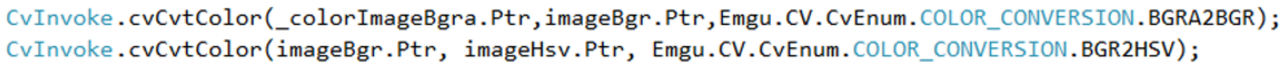
3.2.2 Image Segmentation

After both depth- and color images have been captured, the next step is to process the image and isolate the 2D marker position by regional image segmentation as described in 2.2.1. *OpenCV* functions are used for this purpose. As this implementation is written in *C#*, *EmguCV* is used, which is a wrapper for OpenCV.

As described in the theory, the optimal image format for segmentation by color is the HSV format. The captured images are in a BGRA format. *BGR* represents the Blue Green and Red channels of the picture while *A* represents the opacity of the pixel. There exist methods in OpenCV for the exact purpose of remapping the image format from BGRA to HSV via BGR.

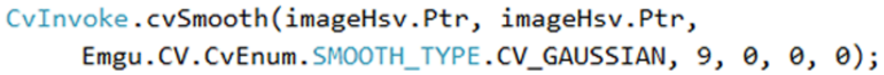
The conversion method accepts a pointer as the first parameter, which points to the memory location of the unconverted image. The pointer for the converted image location is the second parameter while the conversion type is the third. There is no method to directly convert the image format from BGRA to HSV. Because of this, the conversion process will be done in two steps, first from BGRA to BGR, and then from BGR to HSV.

The relevant functions are shown below, where *imageBgr*, *imageHsv* and *colorimageBGRA* are EmguCV image objects. The pointer for the memory location of the image can be read from the image object, by using the *Image.Ptr* variable.



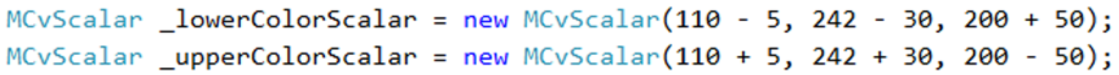
```
CvInvoke.cvCvtColor(_colorImageBgra.Ptr,imageBgr.Ptr,Emgu.CV.CvEnum.COLOR_CONVERSION.BGRA2BGR);
CvInvoke.cvCvtColor(imageBgr.Ptr, imageHsv.Ptr, Emgu.CV.CvEnum.COLOR_CONVERSION.BGR2HSV);
```

After converting the images to an HSV format, they can be smoothed by a Gaussian filter, as described in 2.2.1. However, this is computationally heavy and will slow the frame rate down. As the images in this implementation contain low amounts of noise, the Gaussian smoothing will be obsolete for many scenes.



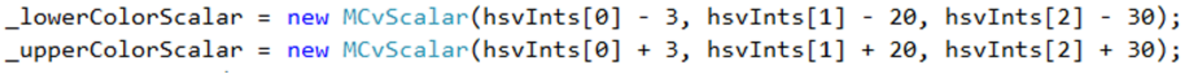
```
CvInvoke.cvSmooth(imageHsv.Ptr, imageHsv.Ptr,
    Emgu.CV.CvEnum.SMOOTH_TYPE.CV_GAUSSIAN, 9, 0, 0, 0);
```

Regardless of if smoothing takes place or not, the image needs to be thresholded. This is done by declaring two objects of the MCvScalar type. Each McvScalar holds the info of one color, as it contains a value for each channel. The excerpt below shows the implementation of these scalars, where their arguments represent the hue, saturation and value respectively.



```
MCvScalar _lowerColorScalar = new MCvScalar(110 - 5, 242 - 30, 200 + 50);
MCvScalar _upperColorScalar = new MCvScalar(110 + 5, 242 + 30, 200 - 50);
```

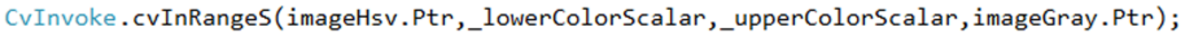
The code above demonstrates the initialization of the upper and lower thresholds. They are initialized with an arbitrary color. When selecting the color of the marker by clicking it in the GUI, the upper and lower scalars will be based on the color sampled at the origin of the mouse click. There will be a small predetermined interval for the three values where a narrow pass-band will be applied for the hue while a wider passband will be used for the saturation and value. This can be adjusted according to the environment and scene for optimal segmentation.



```
_lowerColorScalar = new MCvScalar(hsvInts[0] - 3, hsvInts[1] - 20, hsvInts[2] - 30);
_upperColorScalar = new MCvScalar(hsvInts[0] + 3, hsvInts[1] + 20, hsvInts[2] + 30);
```

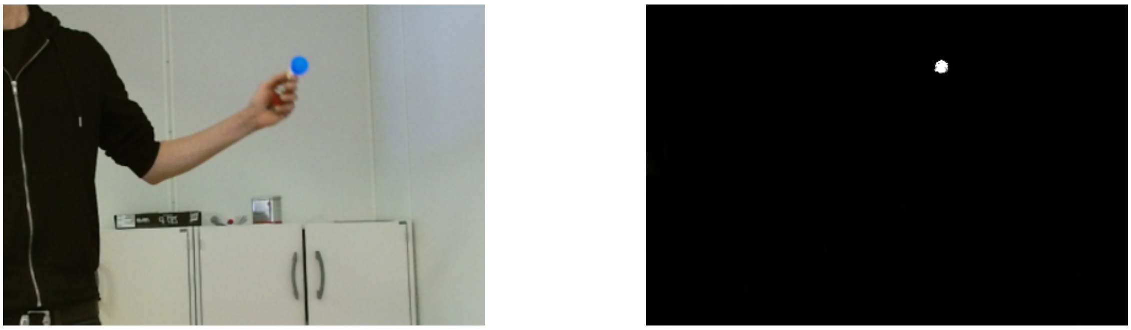
The code above is an excerpt of the code where the threshold levels are redefined. *hsvInts[]* are an array of the HSV values collected from the color image by selecting the desired color to be tracked from the GUI.

The *InRange(...)* method is used for the thresholding and requires pointers to the source and destination images, as well as the upper and lower thresholds, as defined by the scalars. The source is the HSV image, while the destination is a designated memory allocation, as referenced by the pointer.



```
CvInvoke.cvInRangeS(imageHsv.Ptr,_lowerColorScalar,_upperColorScalar,imageGray.Ptr);
```

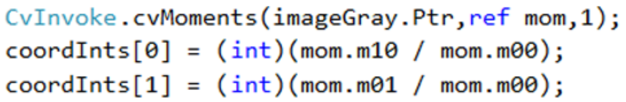
After the segmentation, a binary image remains, where the marker is indicated by the true pixels, displayed as white in the image.



(a) Marker before thresholding *(b) Marker after thresholding*

Figure 16: Illustration of the segmentation process.

The thresholding is now complete, and the center of the *True* pixels must be determined. As described in 2.2.1, the image moments are used for this purpose. OpenCV has the functionality to calculate the image moments in a grayscale image. By supplying a pointer for the segmented binary image, the *CvMoments(...)* method will acquire the image moments needed, and store them in a referenced variable. The segment of code below illustrates this.

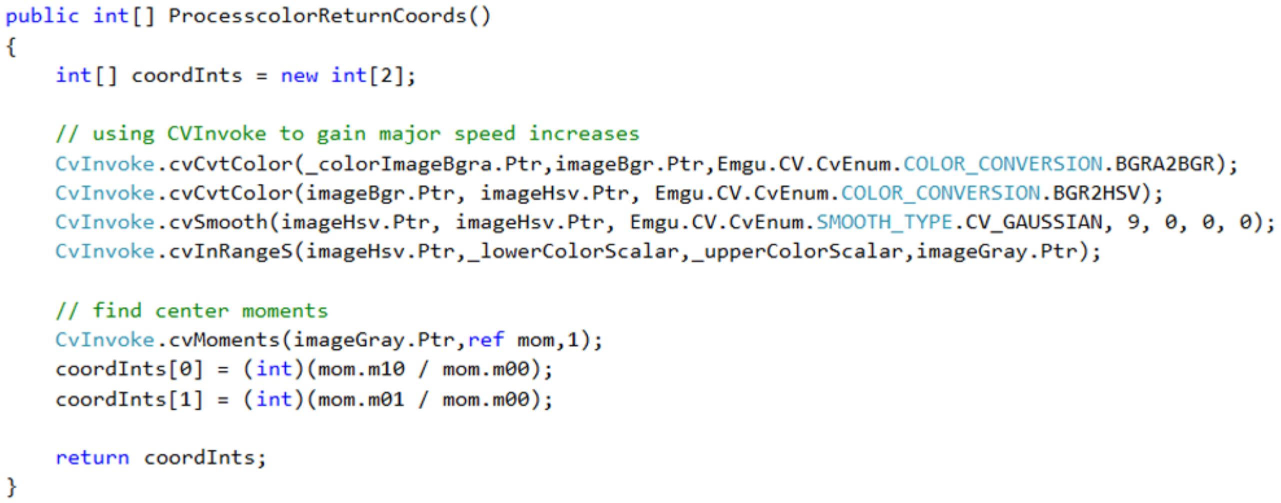


```
CvInvoke.cvMoments(imageGray.Ptr,ref mom,1);
coordInts[0] = (int)(mom.m10 / mom.m00);
coordInts[1] = (int)(mom.m01 / mom.m00);
```

The center moments are calculated according to equation 5, where the resulting coordinates are stored in an array representing the 2D coordinates of the marker, as seen from the perspective of the color camera. Before the 3D coordinates can be determined, the 2D coordinates must be mapped to its corresponding coordinates in the depth image, to compensate for the offset between the color camera and depth sensor.

The image segmentation section of the software is the most computationally demanding. Because of this, the OpenCV methods are run in a separate, asynchronous thread. This prevents the GUI, and other functionality from freezing.

Another significant aspect of these methods are the use of pointers and fixed memory locations. By manipulating the image at its memory location, instead of copying it to a new object, the total processing time dropped significantly. To keep up with the 30 fps capture rate of the Kinect, the processing time should be kept below $30/1000 \approx= 33ms$. By utilizing pointers, where OpenCV works directly on the memory, the processing time got the segmentation dropped from $\sim 60ms$ to $\sim 20ms$. This is inside the timeframe that will allow the application to run in real-time. The full segmentation method is shown below and represents the image processing methods used.



```
public int[] ProcesscolorReturnCoords()
{
    int[] coordInts = new int[2];

    // using CVInvoke to gain major speed increases
    CvInvoke.cvCvtColor(_colorImageBgra.Ptr,imageBgr.Ptr,Emgu.CV.CvEnum.COLOR_CONVERSION.BGRA2BGR);
    CvInvoke.cvCvtColor(imageBgr.Ptr, imageHsv.Ptr, Emgu.CV.CvEnum.COLOR_CONVERSION.BGR2HSV);
    CvInvoke.cvSmooth(imageHsv.Ptr, imageHsv.Ptr, Emgu.CV.CvEnum.SMOOTH_TYPE.CV_GAUSSIAN, 9, 0, 0, 0);
    CvInvoke.cvInRangeS(imageHsv.Ptr,_lowerColorScalar,_upperColorScalar,imageGray.Ptr);

    // find center moments
    CvInvoke.cvMoments(imageGray.Ptr,ref mom,1);
    coordInts[0] = (int)(mom.m10 / mom.m00);
    coordInts[1] = (int)(mom.m01 / mom.m00);

    return coordInts;
}
```

3.2.3 Coordinate Mapping

The 2D position of the marker has been found, but its corresponding location in the depth image is still unknown. The Kinect device is factory calibrated and can map points between its coordinate spaces.

The Kinect features three coordinate systems, the camera space, depth space, and color space. The camera space is centered around the depth measurement sensor and is the default coordinate space. As the figure below (17) shows, the origin of the camera space is at the center of the IR sensor while the Y axis is vertical, and X is horizontal from right to left. In camera space, the units are measured in meters.

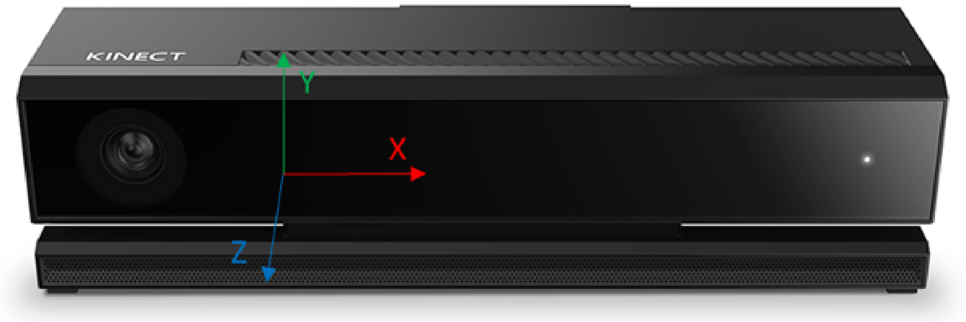


Figure 17: Illustration of the Kinect camera space coordinate system. [18]

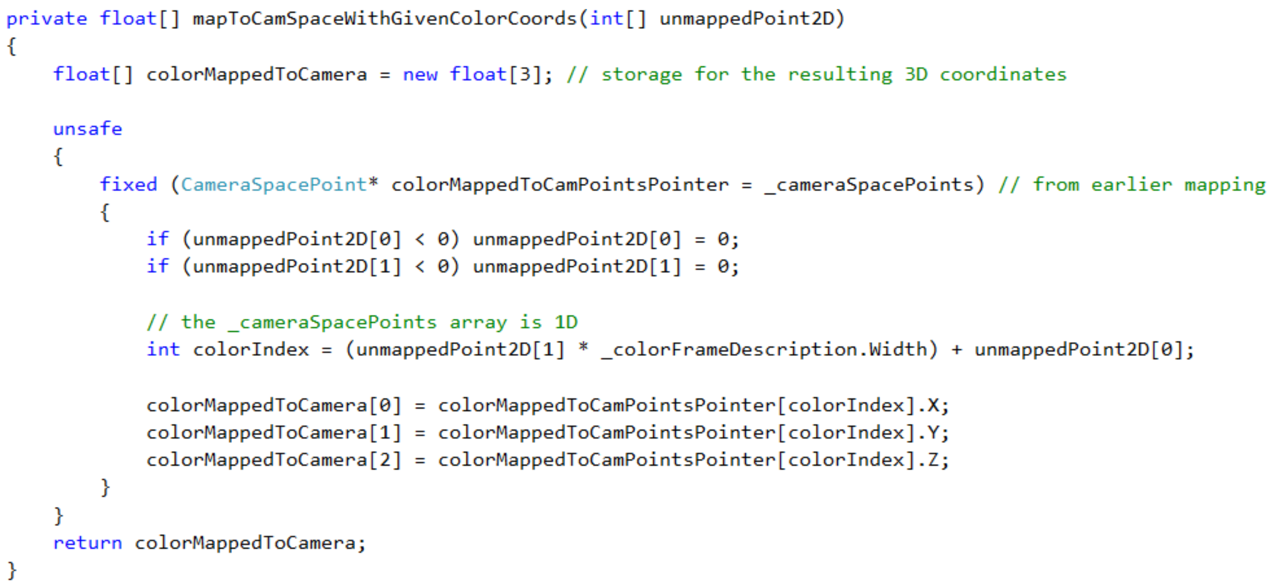
The depth space has the same origin as the camera but is a description of the captured depth image. Color space is the last coordinate system and is offset from the camera- and depth space, as the color camera and depth sensor are placed at different locations on the Kinect device.

In the case of this implementation, the color images are captured and analyzed, which results in coordinates in the color space. To find their corresponding location in the depth image, the color coordinates can be mapped to the camera space.

As each sensor comes factory-calibrated, the intrinsic parameters of each sensor are stored within it. These parameters can be accessed by methods in the Kinect SDK library, which can also do the mapping between the spaces. By first creating an empty CameraSpacePoints[ ] array to store the map data, the MapColorFrameToCameraSpaceUsingIntPtr(...) method can be used. This method uses the depth frame data to map the entire frame captured from color space to camera space. As a relation between the two color spaces has been established, the coordinates from the color image can be used to probe the depth image for its depth in its correct location. This is computationally intensive, as the entire scene must be mapped. Mapping from depth to color is simpler, as the depth is known before the mapping.

The segment of code below shows how the 2D coordinates of the unmapped color space are used to extract the 3D coordinate of the corresponding point in camera space.

[18] https://msdn.microsoft.com/en-us/library/dn785530.aspx



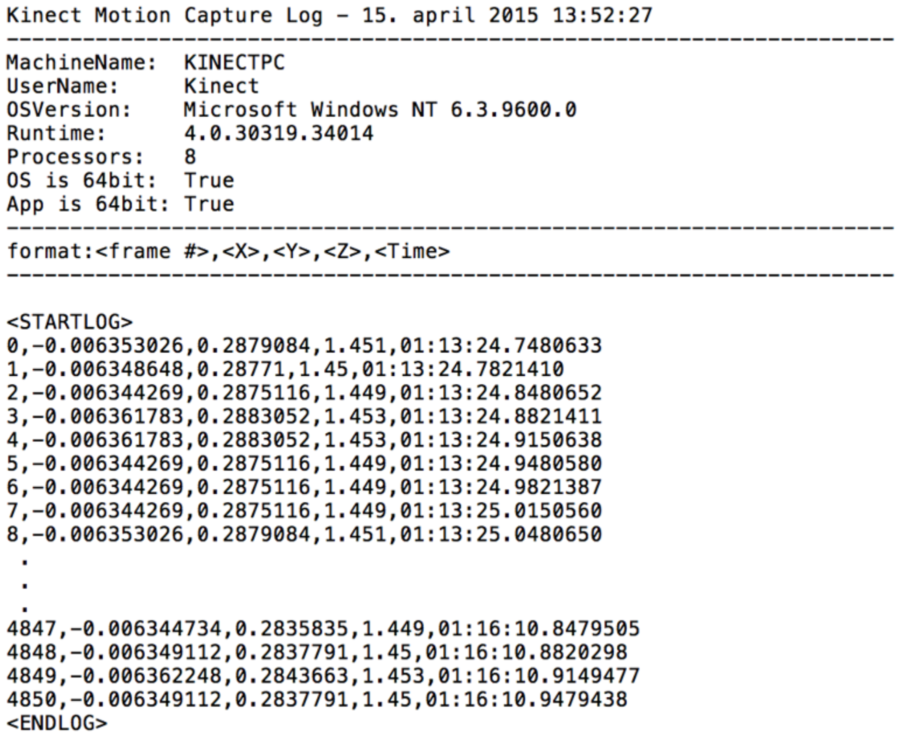
```
private float[] mapToCamSpaceWithGivenColorCoords(int[] unmappedPoint2D)
{
    float[] colorMappedToCamera = new float[3]; // storage for the resulting 3D coordinates

    unsafe
    {
        fixed (CameraSpacePoint* colorMappedToCamPointsPointer = _cameraSpacePoints) // from earlier mapping
        {
            if (unmappedPoint2D[0] < 0) unmappedPoint2D[0] = 0;
            if (unmappedPoint2D[1] < 0) unmappedPoint2D[1] = 0;

            // the _cameraSpacePoints array is 1D
            int colorIndex = (unmappedPoint2D[1] * _colorFrameDescription.Width) + unmappedPoint2D[0];

            colorMappedToCamera[0] = colorMappedToCamPointsPointer[colorIndex].X;
            colorMappedToCamera[1] = colorMappedToCamPointsPointer[colorIndex].Y;
            colorMappedToCamera[2] = colorMappedToCamPointsPointer[colorIndex].Z;
        }
    }
    return colorMappedToCamera;
}
```

As the camera space points are stored in a 1D array, the position of the desired pixel must first be determined.

3.2.4 Data Logging

The 3D positions of the marker through the time of the movement are now known, and can be further processed if wanted. To effectively allow for easy processing by other software, a log file will be created. The recorded coordinate points will be logged to a text file, which will be written at the end of the recording session.

The log file will have a specific, consistent structure and can easily be parsed by other software. After a header, describing general info of the recording system, the data will be presented in a CSV (Comma Separated Value) format, as illustrated in the figure below.



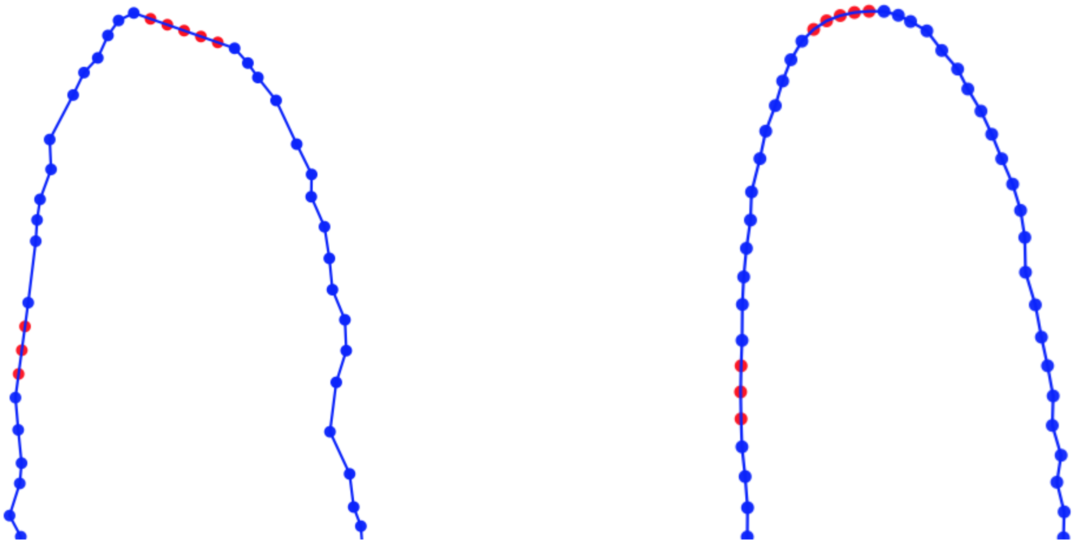
```
Kinect Motion Capture Log - 15. april 2015 13:52:27
----------------------------------------------------------------------
MachineName:   KINECTPC
UserName:      Kinect
OSVersion:     Microsoft Windows NT 6.3.9600.0
Runtime:       4.0.30319.34014
Processors:    8
OS is 64bit:   True
App is 64bit:  True
----------------------------------------------------------------------
format:<frame #>,<X>,<Y>,<Z>,<Time>
----------------------------------------------------------------------

<STARTLOG>
0,-0.006353026,0.2879084,1.451,01:13:24.7480633
1,-0.006348648,0.28771,1.45,01:13:24.7821410
2,-0.006344269,0.2875116,1.449,01:13:24.8480652
3,-0.006361783,0.2883052,1.453,01:13:24.8821411
4,-0.006361783,0.2883052,1.453,01:13:24.9150638
5,-0.006344269,0.2875116,1.449,01:13:24.9480580
6,-0.006344269,0.2875116,1.449,01:13:24.9821387
7,-0.006344269,0.2875116,1.449,01:13:25.0150560
8,-0.006353026,0.2879084,1.451,01:13:25.0480650
 .
 .
 .
4847,-0.006344734,0.2835835,1.449,01:16:10.8479505
4848,-0.006349112,0.2837791,1.45,01:16:10.8820298
4849,-0.006362248,0.2843663,1.453,01:16:10.9149477
4850,-0.006349112,0.2837791,1.45,01:16:10.9479438
<ENDLOG>
```

3.2.5 Data Processing

To process the logged data, a Python script has been written. The purpose of this script is to filter the signal and remove possible outliers and errors.

The first action taken by this script is removing the outliers. Outliers are created in some cases when the coordinates calculated from the color image misses the location of the marker, and a faulty depth is sampled. This can be caused by inaccuracies in the image processing, such as noise in the image, or too rapid motion of the marker for the image processing or depth sensor to follow. Another source is simply occlusion of the marker for any period of time. Errors in measurements such as these will often yield a significant change in position. They can because of this be easily classified as outliers and excluded from the data set.

Excluding points in the data set, for any reason, leaves gaps in the path. If these are only an occasional small gap, it can be bridged. An algorithm written for this purpose will remember the last valid point as it iterates through the path. If the next point is marked as invalid, as it is classified as an outlier, it will start the bridging sequence. The algorithm will count the invalid points until it reaches a valid one. Every invalid point in the segment will be recreated. The points will be distributed evenly between the two valid points in either end. This will only give a realistic approximation to the original path if the gap is short, which it would be if it were caused by noise, etc. However, if the invalid points were caused by occlusion, the gap will be larger, and this approach may not leave a realistic reconstruction of the path.



(a) Bridge (red) created in gap where no positional data exists

(b) Path after moving average filter (window size = 3)

Figure 18: Illustration of the bridging process, for gaps in recorded data

After bridging eventual gaps, the signal will be filtered through a lowpass, boxcar filter also known as a moving average filter, as described in 2.4. Filtering will remove a large number of disturbances in the signal, depending on the window size. A large window size will reduce much of the noise, but will also suppress any sudden, but intended movements, such as a sharp corner.

3.3 Hardware Implementation

Most of this implementation are done through the Kinect device, and by software, where not much consideration needs to be taken regarding hardware. Had this been a stereo vision system, the hardware configuration of the system would be of much higher importance, as extrinsic parameters play a crucial role (see 2.2.2).

The software runs on an Intel i5 3.5GHz processor, with the GeForce GTX 750 GPU. The Kinect interfaces with the PC through USB3.

3.3.1 Marker

One highly hardware dependent aspect of this system is the marker. Simple experiments have been made to determine the most important features of a good marker. As the Kinect will use a color image to determine the location of the marker, one of the weak points of the system is the thresholding phase.

The marker should easily distinguish itself from the scene by its color, which should be as uniform as possible, to make the entire marker fall within the pass band of the threshold more easily. One of the main factors determining if an object can be used as a suitable marker is its gloss. If the marker is glossy, it will reflect light from the scene instead of its color. This means an object with a shiny surface is poorly suited as a marker. Diffuse or ”frosted” markers are much more suitable. By having a diffuse surface, the light will scatter in all directions, resulting in a smoother and more uniform surface. For this implementation, a ping-pong ball with a blue color was chosen. As it was of a manageable size and relatively diffuse, it served well as a marker. To further highlight the color and thereby increase robustness towards varying light conditions, a small LED was placed in the ball.

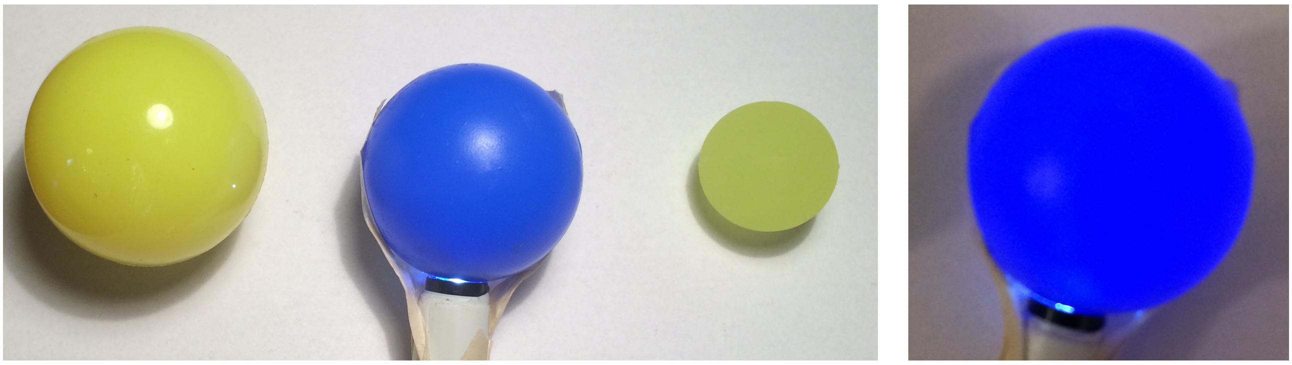


Figure 19: Different markers displayed, where the ball to the far left is the most glossy. The blue ping-pong ball is illuminated and somewhat glossy while the smaller ball is matte. The ping-pong ball in a darker environment is illustrated to the far right and shows a highly uniform color.

An alternative approach to the colored marker was also tried, where an infrared marker was used instead. The IR marker position would be captured directly by the depth measurement sensor, and would not require mapping. However, the IR diode interfered with the time-of-flight technology and resulted in an infinite depth at the location of the marker.

4 Experiments and Results

To determine the feasibility of using the Kinect as a motion capture sensor, some tests will be performed. The core purpose of these tests is ascertaining the performance of the Kinect as a measurement device. Two categories of tests will be conducted, static and dynamic. The static tests will focus on the precision and accuracy of the provided measurement, in a static environment. The focus of dynamic tests is to determine if these measurements will also apply for a moving marker, at various velocities.

4.1 Static Tests

The purpose of the static test is to determine the accuracy and precision of the measurements provided by the Kinect. The camera aspect, which will yield the 2D coordinates, will be examined, but the most interesting part of the Kinect technology is its depth measurement capabilities. As the 3D location is heavily dependent on the depth measurements, this will be closer studied.

4.1.1 Setup

The setup of this test is relatively straightforward. The Kinect will be placed at a permanent location while the marker will be placed at various predetermined locations in front of the Kinect. The distance between the Kinect and marker is manually measured by measuring tape and compared to the measurements given by the Kinect.

There are some sources of error in this setup, one of the largest might be the manual measurement of the distance, which will affect the accuracy of the measurements. Another source of error is determining the origin of the range measurement. As the camera space coordinate system starts at the center of the depth measurement sensor, inside the Kinect, its location is hard to pinpoint.

4.1.2 Results

Despite the sources of errors for this test, such as manual error measurement, and an eventual bias caused by an undetermined source of origin, the results should ideally give valuable information regarding the precision, accuracy and possible distortions.

As the origin of the camera space is hard to determine accurately, it will naturally result in a systematic bias in depth measurement. The bias can easily be compensated for, as a new point of origin can be defined in the camera space. Every point captured after this can now be described relative to this new reference point, leaving any systematic bias obsolete. As bias can easily be compensated for, it is important to establish if there is any form of scaling error or non-linear error. Even if a point of origin is determined, the captured positions may contain an error in accuracy related to the distance of the point. This relation may be linear, or non-linear. This error can also be compensated for if correctly modeled.

The ideal results of this test will be data indicating a close match between the measured and actual distance, with a low variance. Below is a graph describing the measurements of the Kinect compared to the actual distance. A similar test of the Polhemus systems depth measurements is included. The test of the Polhemus system carried out in the same way as for the Kinect, by placing the magnetic field generator at a static location and moving the sensor to actual measured distances.

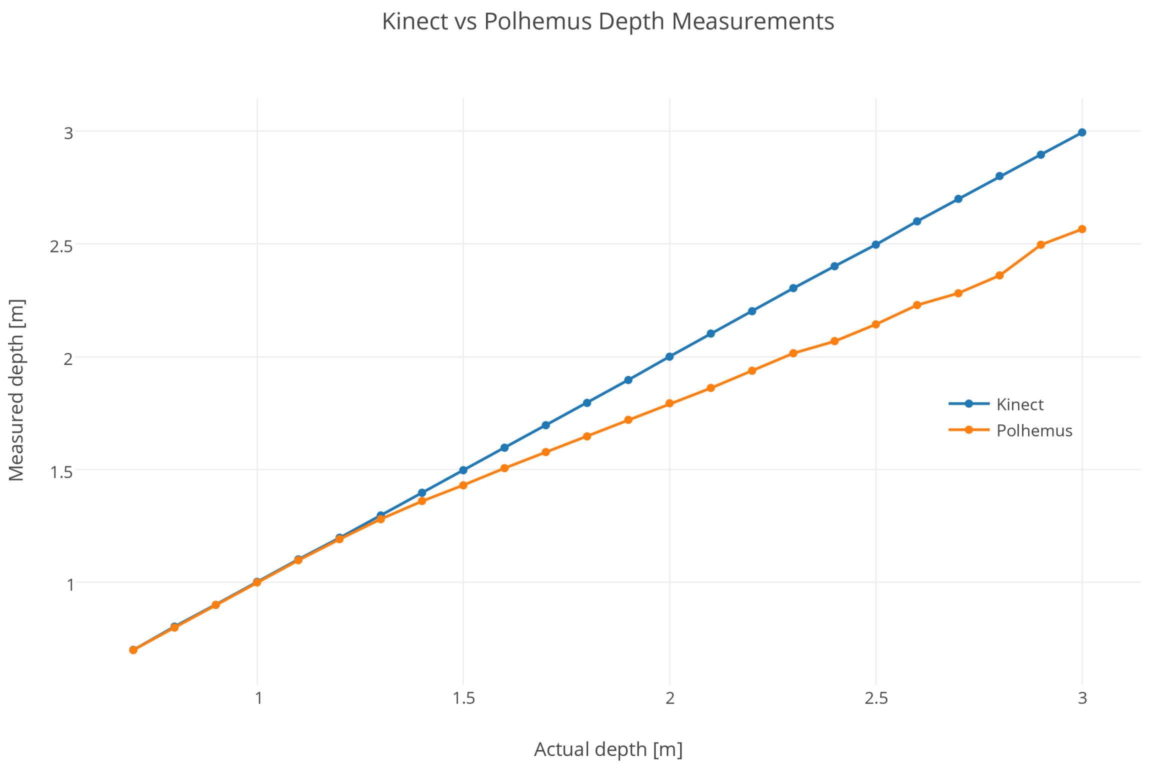
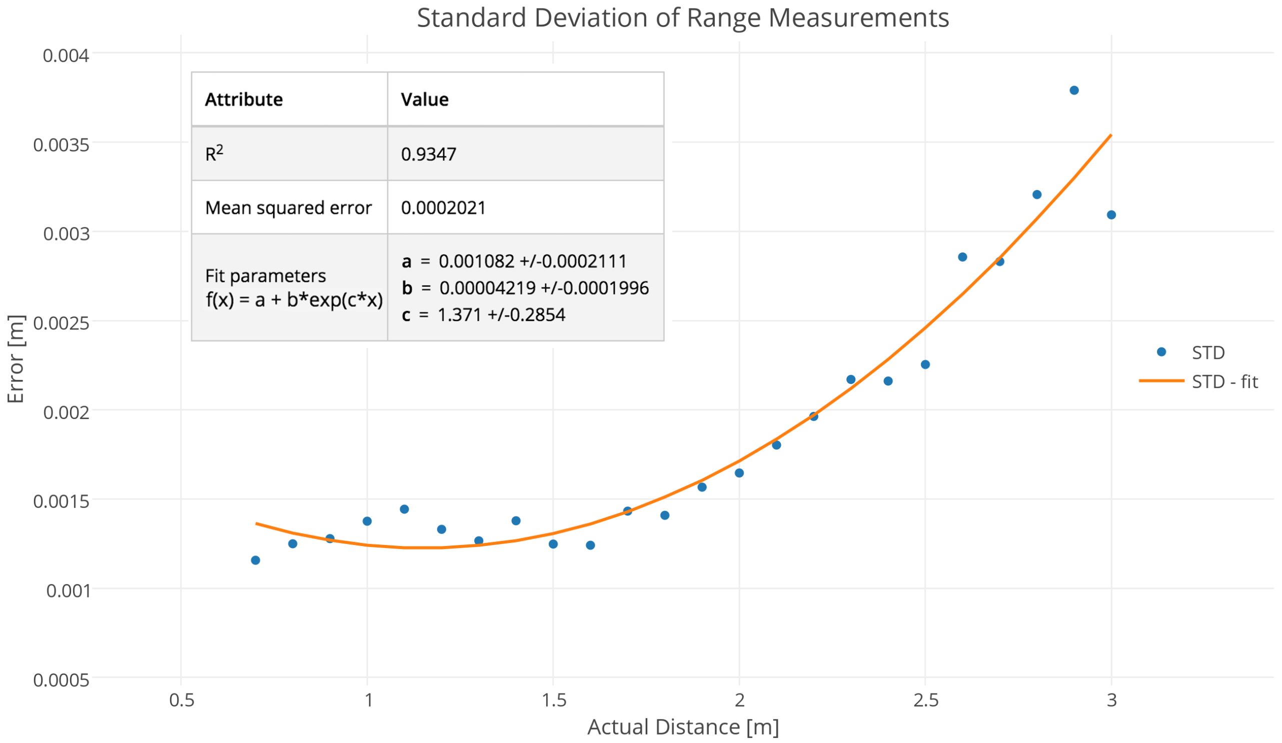


Figure 20: Illustration of the depth measurements of the Kinect and Polhemus, compared to each other.

As figure 20 shows, the measurements were started at a range of 0.6m, as this is approximately the nearest distance the Kinect will work effectively. The observed measurements are practically overlapping until they reach a range of ∼ 1.3m. From this distance, the magnetic sensor starts to deviate from its ideal linear path, by an increasingly large distance. This deviance is expected, as the Polhemus technology relies on magnetic fields, which will dissipate at a distance.

The Kinect, with its time-of-flight technology, keeps an approximately linear path in the entire span of the measured distances. To examine the error of the Kinect measurements in more detail, the standard deviation at each point has been calculated and plotted, 100 samples were used for each distance.

As figure 21 illustrates, the standard deviations of the Kinect's depth measurements increase with range. However, the variations are quite small compared to the distance at which they are measured. The standard deviations in the ∼ 0.6m to ∼ 1.6m range are between 1mm and 1.5mm. This difference increases with range and gives a standard deviation at ∼ 3.5 mm at the far end of the measured ranges. This error is not a highly significant if the intended application is to capture the human movements involved in spray painting.



Attribute	Value
R^2	0.9347
Mean squared error	0.0002021
Fit parameters f(x) = a + b*exp(c*x)	**a** = 0.001082 +/-0.0002111 **b** = 0.00004219 +/-0.0001996 **c** = 1.371 +/-0.2854

Figure 21: Illustration of the standard deviation in relation to the measured distance.

The standard deviation calculated from this test resembles an exponential function, as shown in the figure, where a fitted exponential function is overlaid the estimated standard deviations. This function indicates the error is increasing at an exponential rate to the distance.

To extend the scope of this experiment, the depth measurement was also tested at different locations along the x-axis, which is the horizontal axis of the Kinect coordinate system. The purpose of this extended test was to create a grid of measurements in the x,z plane.

The results of this test will ideally indicate if the depth measurements will still follow the same general pattern, even if the measurements are not taken along the optical (z) axis. This experiment should also provide data to determine if the camera measurements suffer from any distortions, as the measurements would not follow a linear pattern if this were the case.

Ideal results from this test should show precise depth measurements at all positions in the x-axis not only towards the center (x=0). As the focus is depth, a wide marker ($\sim$ 10cm $\cdot$ 10cm) was used. This size of marker resulted in a non-precise x,y measurement, but a robust depth measurement. The resulting grid can be seen in figure 22

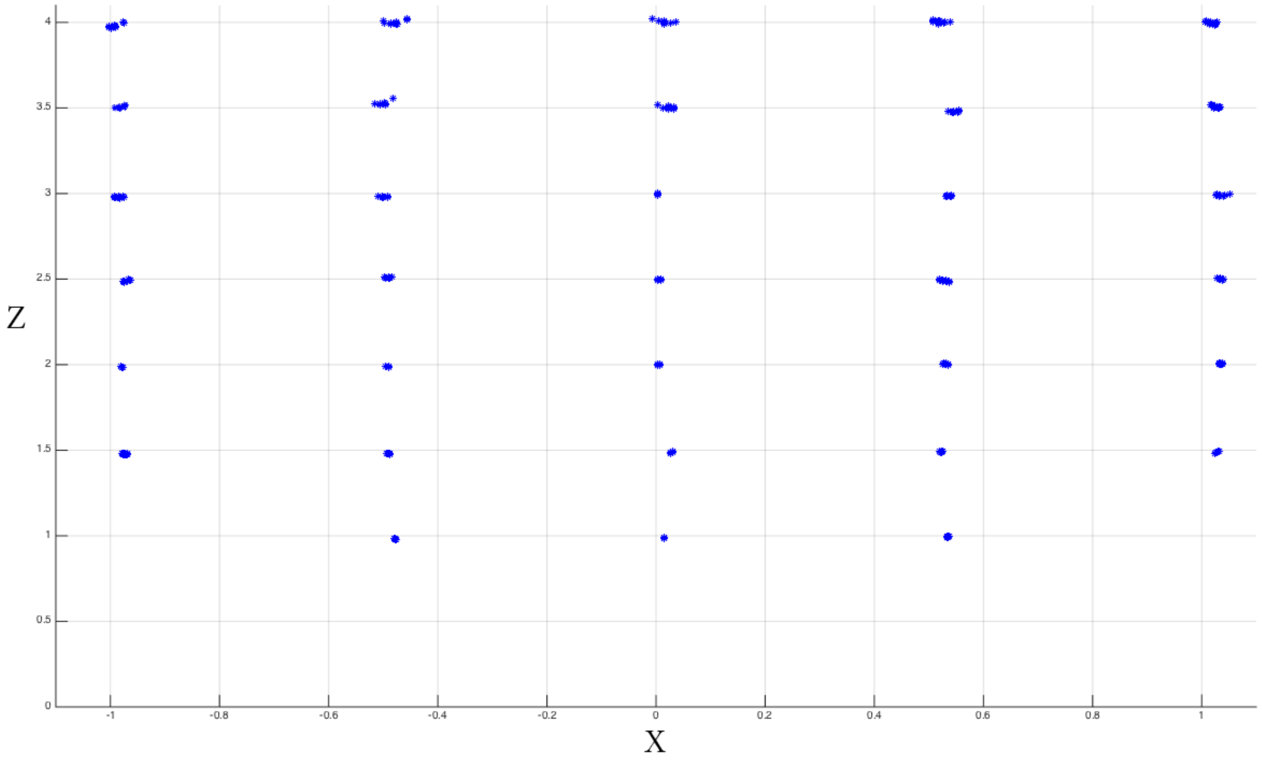


Figure 22: Illustration of the grid produced by x,z measurements.

As figure 22 shows, the depth measurement remained relatively precise at every position. The grid of measurements also shows a straight linear relationship between the clusters of data, indicating a very low presence of lens distortions, if any. Had lens distortion affected the measurements, a non-linear relationship between the clusters could be observed, but as every cluster is positioned in a relatively straight grid this seems not to be the case. As lens distortion does not affect the measurements, it has been correctly compensated for by the factory calibration.

In summary, the observations from the static test indicate an accurate, precise and linear measurement pattern. The system calibration seems to compensate for any eventual sources of systematic error, such as distortion and has so far provided satisfactory measurements for motion capture applications. However, these measurements were performed in a static setting, where disturbances from movement or a smaller marker were not taken into consideration. The dynamic tests will address this.

4.2 Dynamic Tests

The purpose of the dynamic test is to determine the measurement capabilities of the Kinect in an environment where the marker is moving. This test will simulate the intended application of the motion capture system in a controlled environment. To get comparable results, the path will remain unchanged while the velocity will vary in this array of tests.

4.2.1 Setup

To accurately repeat the same path, the marker is mounted to the tool end of a robotic manipulator. The manipulator follows a programmed path, designed specifically for this test. By utilizing a robotic manipulator, this route can be repeated with effectively identical movements. As the human aspect is removed, this setup allows for repeated precise and accurate reference path.

The path is designed to test measurements in all three dimensions. Figure 23 below illustrates the general path used, which is captured by the system at low velocity. The velocity of the manipulator will be set to 0.25 m/s, 0.5 m/s, 0.75 m/s and 1.0 m/s throughout these tests.

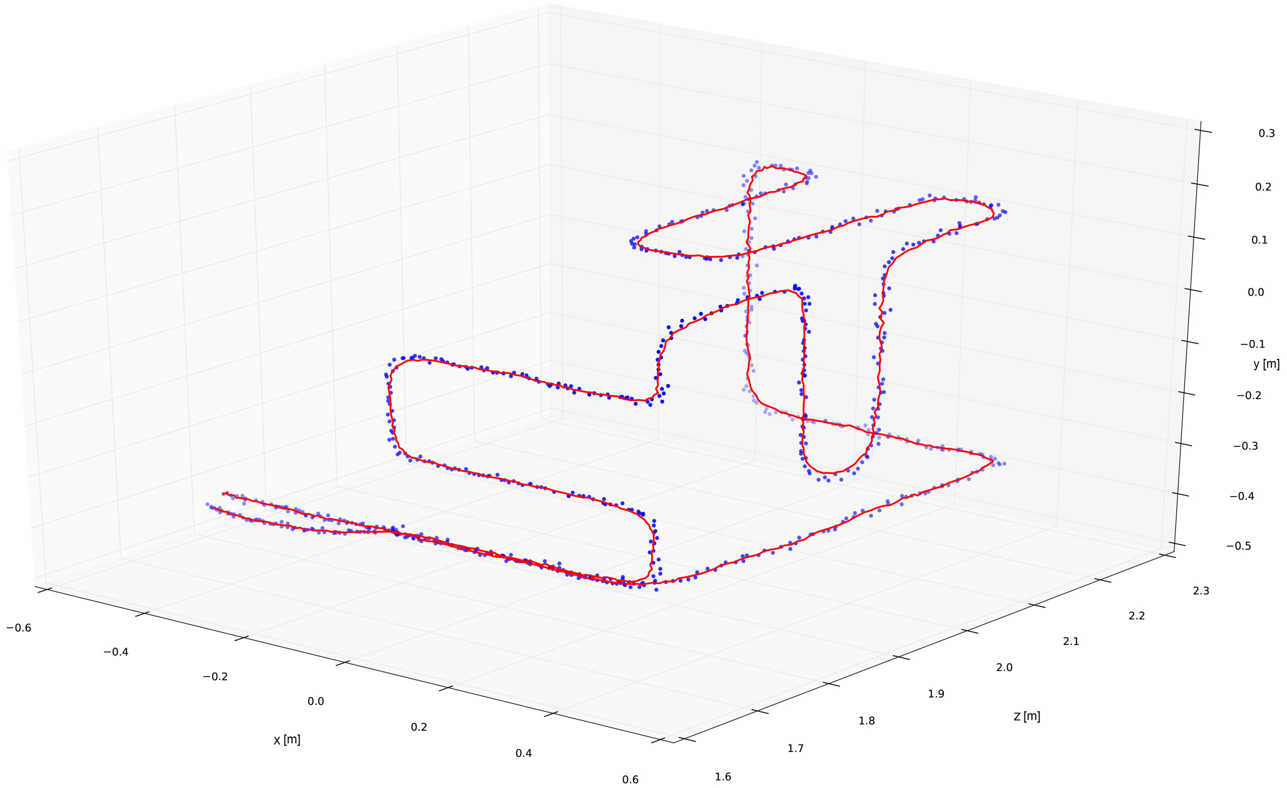


Figure 23: Illustration of the robot path, captured at 0.25 m/s. The captured points are represented by the blue dots, while their moving average is represented by the red line. (Inversed x axis, as the Kinect mirrors the captured image.)

One major source of error in this setup is the lacking measurement of the actual distance from the sensor to the marker. The coordinate system of the sensor is not calibrated against the coordinate system of the robot. This will result in two different coordinate systems, with no reference to each other. However, as range measurements were tested in the static test, the focus of this test will be the relative positions measured in the robot coordinate system, and the relative accuracy and precision of these measured points.

4.2.2 Results

The first step of the dynamic testing will be to capture a number of paths and compare them against each other. A path in this context is a recorded capture of all coordinates sampled in one cycle of the robots program. Before starting to analyze the captured paths, a proper method of comparing paths should be established.

The captured data will consist of discrete sampled points, as the velocity increases, these will naturally be further apart, as there is a constant sampling rate of $\sim$ 30 Hz. For comparing two parts against each other, a natural approach would be to compare the n-th point in the first path to the n-th point in the second path, then iterate through the sequence by increasing n, while accumulating the calculated distance between points. This method would work well if the corresponding points in the sequences had no offset. If there is an offset between the sequences, the distance between the corresponding points will increase according to the offset, for every point in the path. As the focus is the spatial distance between the points, a nearest neighbor approach can be used, where the distance between the two nearest points in each path is used to determine the distance.

This will compensate for an offset of the entire sequence of points in a path, as each point would find its nearest corresponding point in the other path by its euclidean distance. The path needs to be trimmed at each end, as excessive points in one path would otherwise use the distance of the last point in the other path as its reference.

By trimming the path to start at roughly the same coordinates, and applying the nearest neighborhood approach, the error measurements between the paths begins to reduce towards their actual error. Even though the offset in the entire sequence is accounted for, there will still be an offset between points in the two paths. The paths may effectively be overlapping, but the points can still be at a distance from each other, as the sampling intervals are relatively high. The offset will not be greater than the distance traveled between samples, but represents an error, and will accumulate.

A solution to this is measuring the distance from a point in one path, to the closest point on a line segment between the two closest points on the other path. By this approach, the offset between points become irrelevant, as the line segment between them is used instead. This is illustrated on a short section of a path in figure 24.

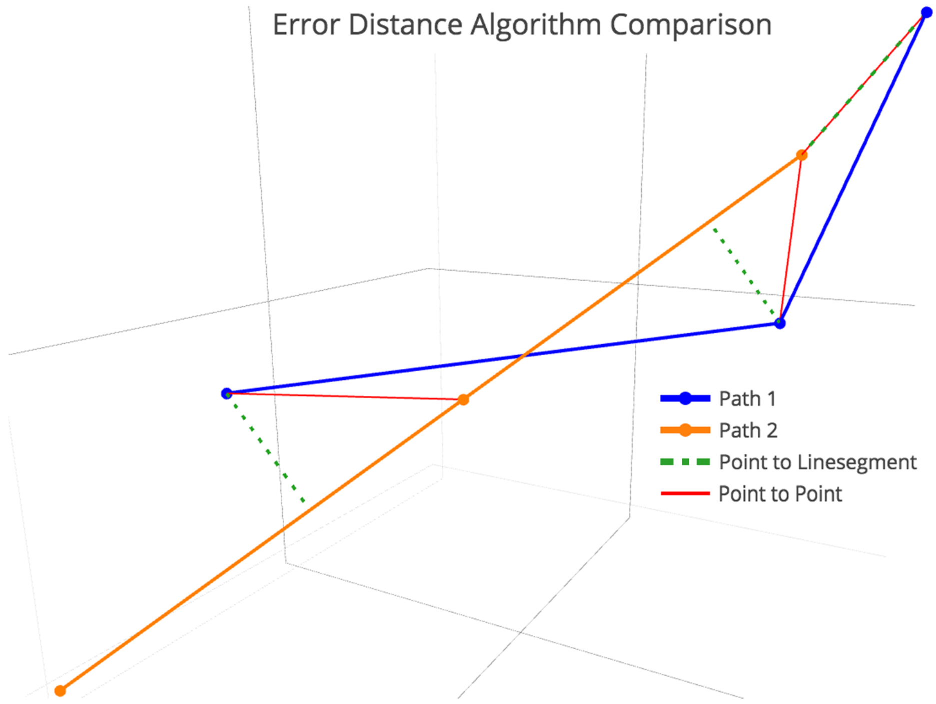


Figure 24: Illustration of two different algorithms, and how distance is measured

As seen in figure 24, the point-to-point approach uses an unnecessary long distance. As the focus of this experiment is to analyze the distance between paths, the point-to-line approach gives a better estimate of the actual distance.

An experiment with these two approaches will show the difference between them. Besides testing the algorithms, this experiment will determine if there is any large change in paths recorded at high and low velocities. The paths will be recorded at 0.25 m/s and 1 m/s, as shown in figure 25.

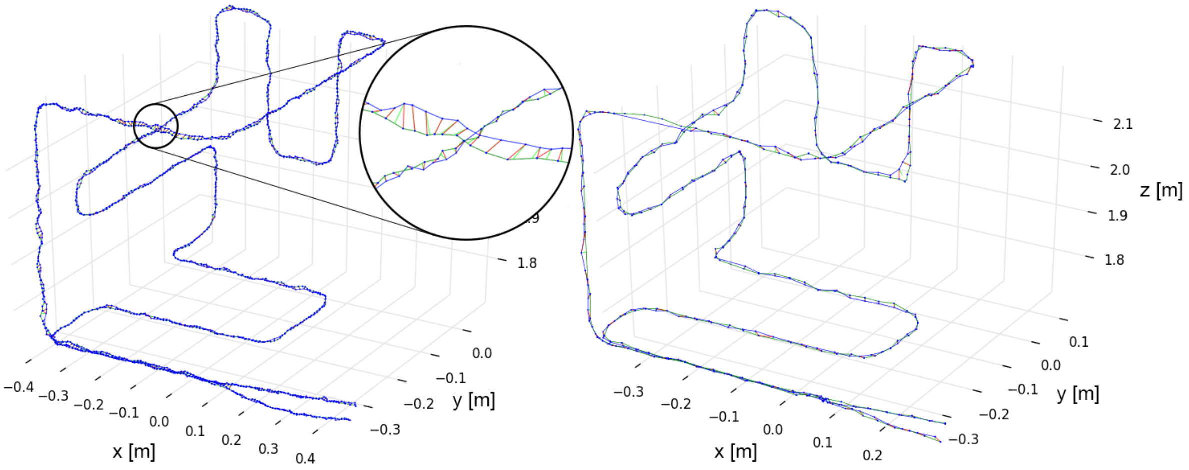


Figure 25: Measurements at 0.25 m/s (left) and 1.00 m/s (right)

The captured paths deviated to a small degree, however only at an average distance of $\sim$ 3.4 mm and 4.7mm for the slow and fast tests respectively. The interesting observation, in this case, is the difference between the two approaches. The increase in error measurement from the point-to-line measurement to the point-to-point measurement was at $\sim$ 31% for the slow test, and *sim* 71 % for the 1 m/s test.

This illustrates the difference between the methods, as the high-speed measurement was more affected by the larger spatial distance between the sampled points, compared to the low-speed test. The point-to-line measurement method is more robust, in this case, as it did not depend on the distance between the sampled points.

Building on these observations, another approach can be applied. Errors arise from an offset between all samples in one path, in relation to the other. A solution can be to shift the entire sequence of points until the sum of least square errors reaches a minimum. This approach will align the paths at its optimal position to each other, but assumes the paths are recorded at the same velocity.

After determining the optimal amount of samples to shift the path, it will be interpolated to increase the number of points in the path. As observed in the algorithm test earlier, a high sample rate will reduce error, as the potential offset between points is minimized.

The next array of tests will examine if the position of the measured points corresponds to its real world equivalents. As the coordinate systems are not calibrated, the positions will be determined relative to the robot path. This test will show if there are any scaling error or disturbances. There will be no measurement of an eventual bias, but as discussed during the static test, a systematic bias can easily be corrected. Scaling and/or other disturbances may be more complex to correct.

For this test the path will be recorded ten times, in four different velocities; 0.25 m/s, 0.5 m/s, 0.75 m/s and 1 m/s. The sequence of average 3D positions of the ten corresponding points from the ten paths will be plotted, and compared to the relative path of the robot. The interesting observations to make regarding these experiments will be to see if the recorded path will travel the full length of the actual path, or if there will be any scaling or distortions. To effectively inspect the recorded positions, the 3D path will be split into its three components, one for x, y and z, as shown in figure 26.

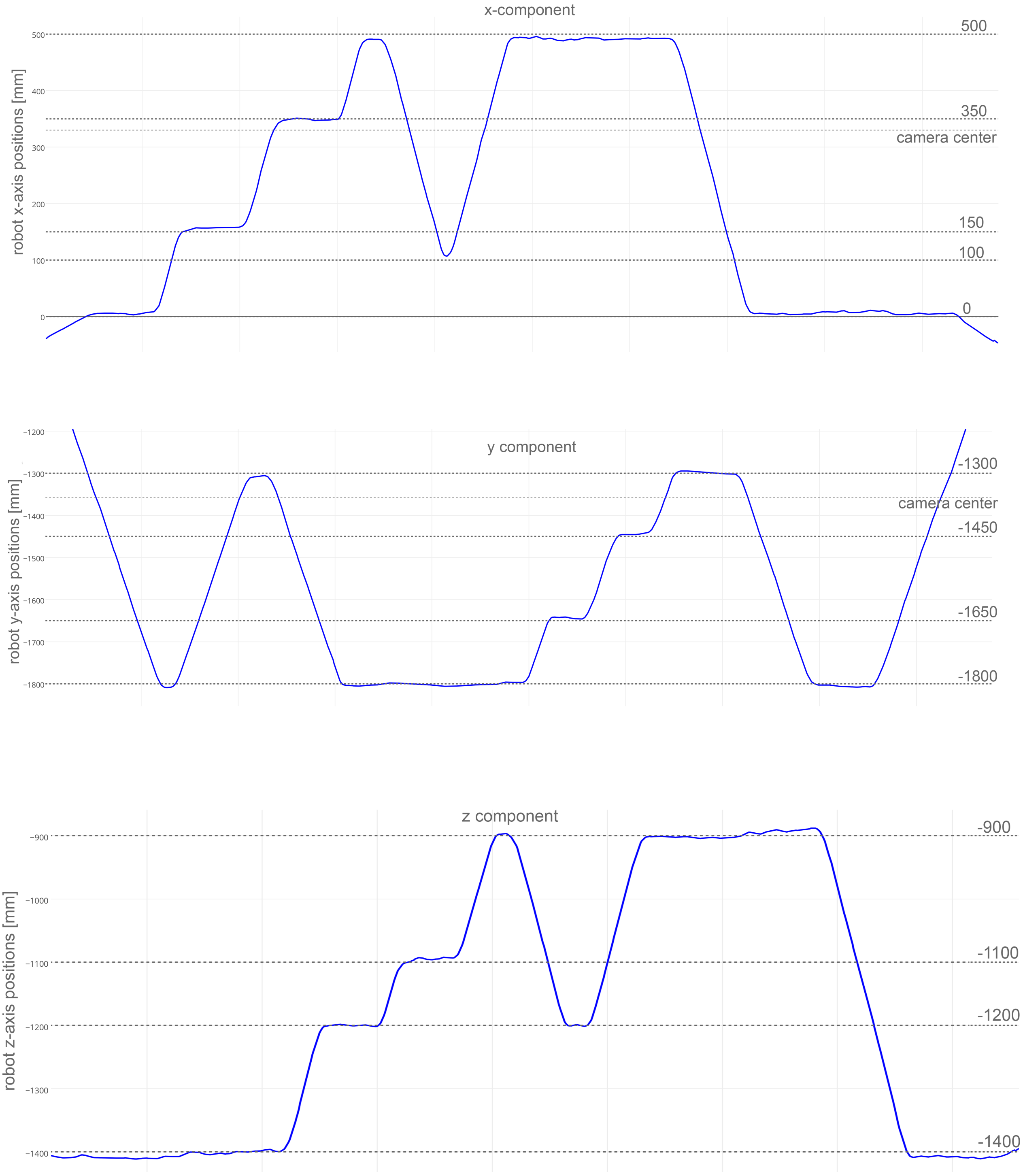


Figure 26: Measurements of the path at 0.25 m/s, with indication of actual positions from the robot path.

These figures show the average position of the marker along the path for each axis. To determine if the positions correspond to the ones intended in the programmed path, their relative position will be compared to those in the program. A dotted line in the plots will indicate the different values used in the original path. At some point in the recorded path, the marker will reach all these positions, and a comparison can be made.

To compare paths, the different positions for each axis in the robot coordinate system were set, and an identical offset was added to each of the points on each axis. The offset was adjusted to the point where the coordinates in the recorded path would fit its corresponding coordinates programmed into the robot.

As mentioned earlier, this test will not take bias into consideration. However, as the Kinect is placed parallel to the robot's coordinate system, this bias can be compensated for as there is no difference in orientation to consider.

By examining the path and its predicted positions from the programmed path, any error related to scaling could be directly observed. From the observations, the measured and predicted positions match as expected, as they effectively overlap, with only a few millimeters in standard deviation.

Some repeated errors are observed in the x and y components. The measured positions are closer to the center of the scene, compared to what the actual path should be. This is not much but does not seem to be random. A possible explanation for this is the geometry of the marker. As the marker is a ping-pong ball, the radius of the sphere will affect the measurements. When the ping pong ball is measured directly in front of the sensor, there will only be a difference in depth measurement, as the marker is considered as a small theoretical point. Once the marker moves to the edges of either the x or y-axis, the measured point will be closer to the center, as the surface of the ping-pong ball is measured, not the center. As this will only account for an error equal to the radius of the ping-pong ball at most, it will be relatively insignificant and can be compensated for as it represents a systematic error.

The recorded positions align well with the estimated positions, and besides the marker surface error, there seem to be no significant scaling or distortions present. This test successfully provided data describing the relative accuracy of the recorded path. The purpose of the next test will be to determine the precision of the recorded data, over a range of different velocities.

Ten cycles of the path will be recorded at four different velocities, and compared against each other. This will determine the extent of the deviation between the captured paths, and if a higher velocity will produce a larger error.

The figure below 27 illustrates the mean path, and the standard deviation relative to the mean path. For the slowest velocity, 0.25 m/s, there is hardly any error between the recorded paths. The average standard deviation per point are shown in figure 31. For practical use in pain applications, an error of this size should not cause any major problems.

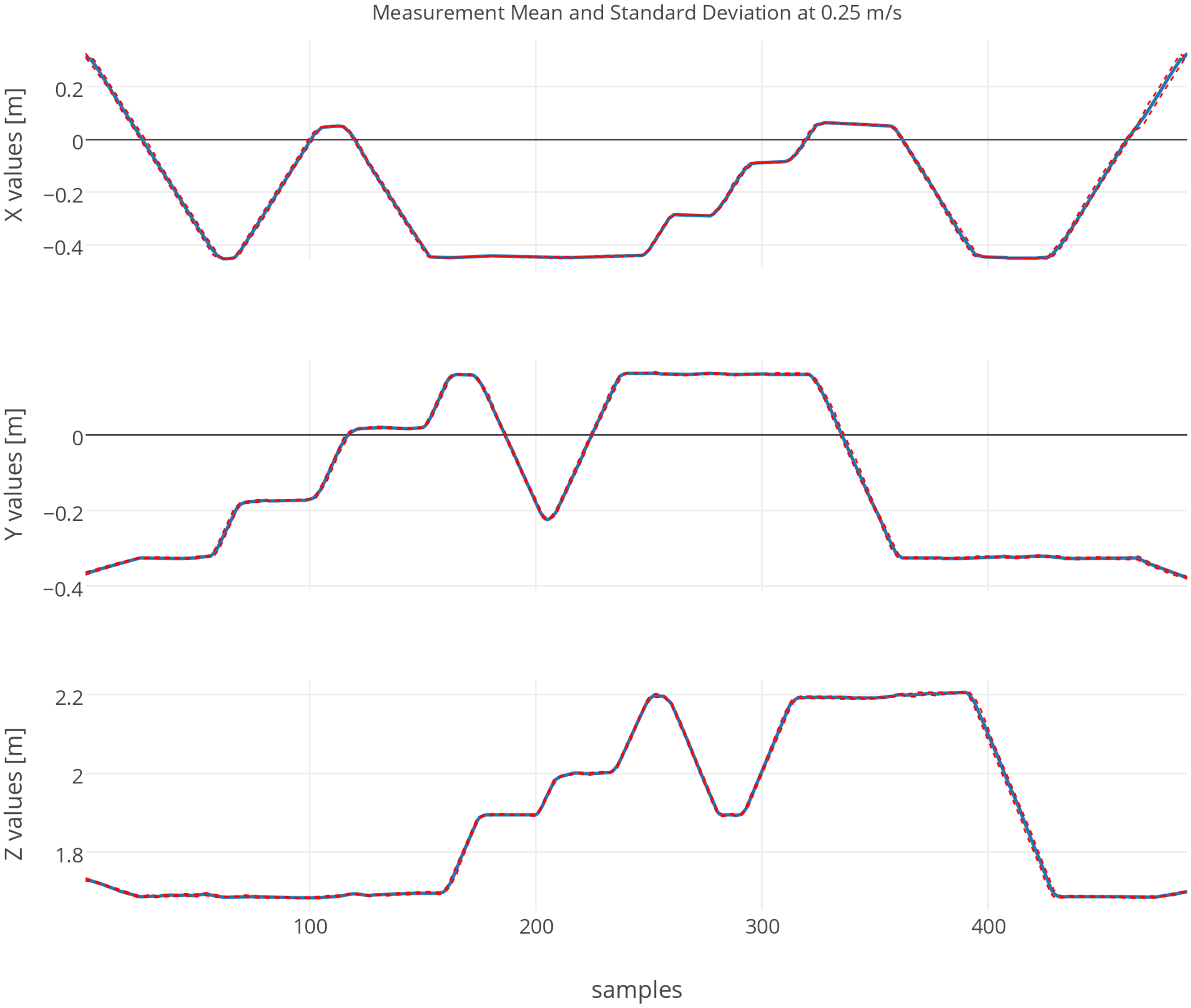


Figure 27: Mean and one standard deviation at 0.25 m/s

Figure 28 illustrates the same path, only at a higher velocity (0.5 m/s)

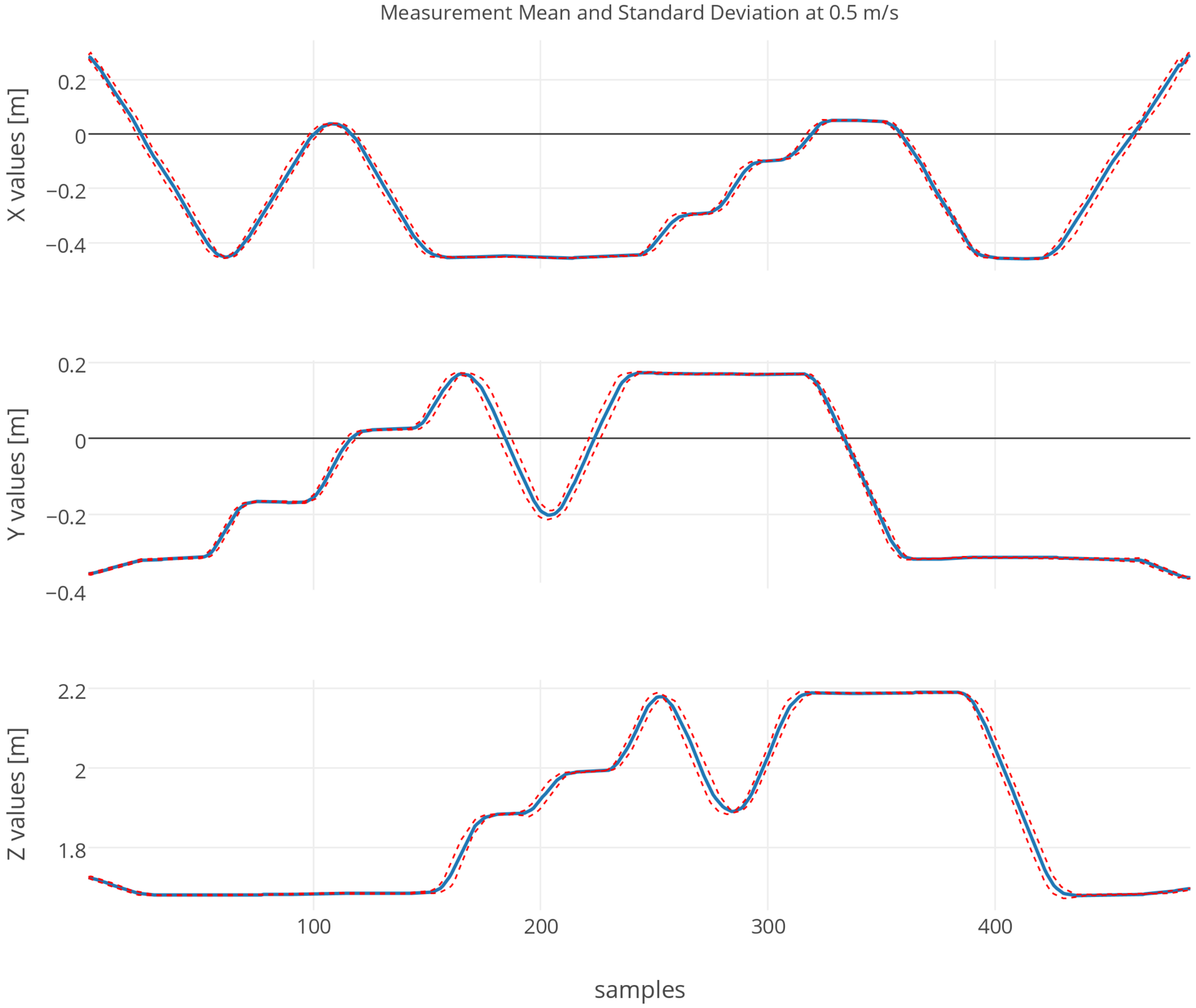
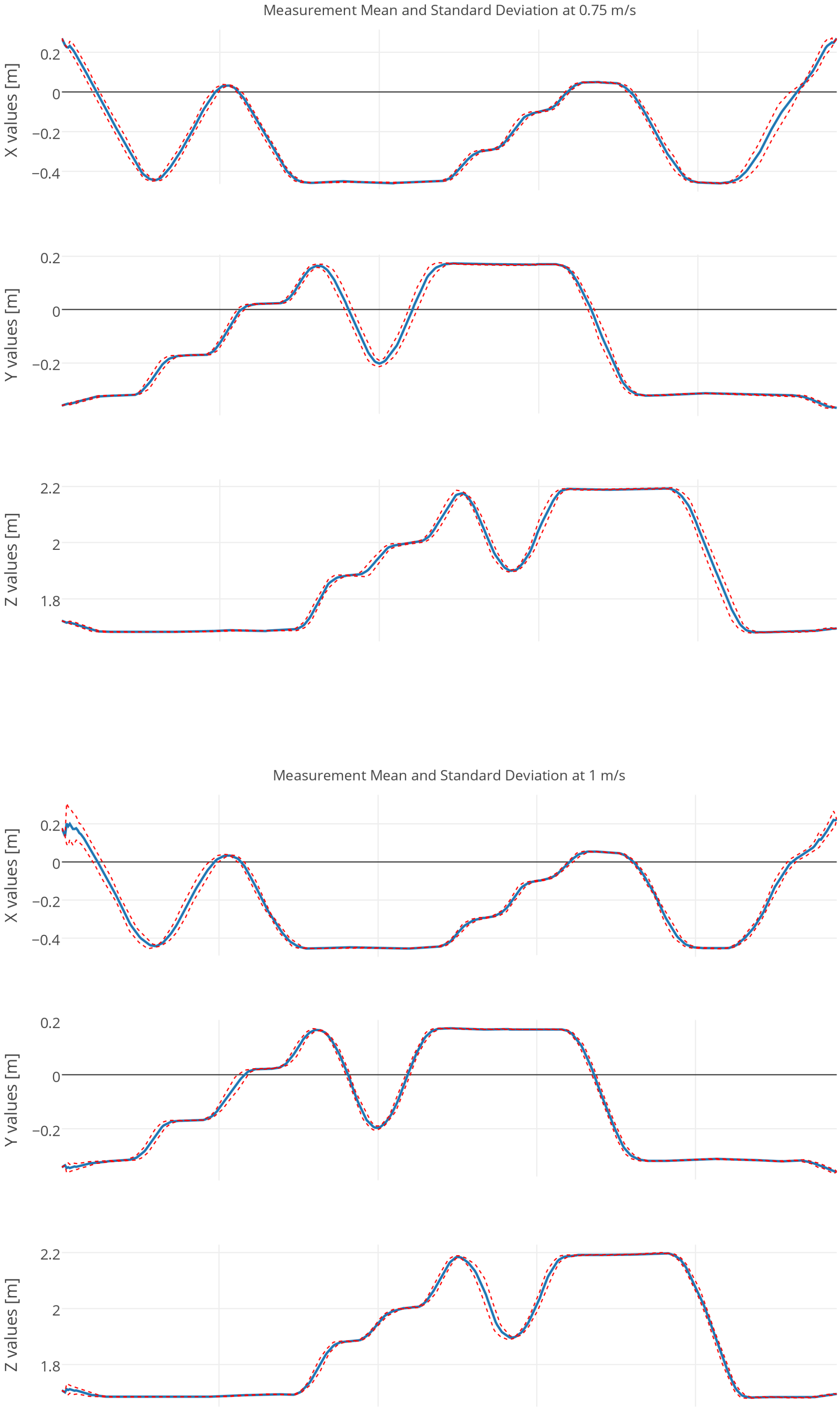


Figure 28: Mean and one standard deviation.

As observed from figure 28, there is a larger error at 0.5 m/s compared to 0.25 m/s. Most of the error occurs while the marker changes position. As expected, when the marker changes position in a given direction, the error in this dimension increases while the movements occur. A probable reason for this might be motion blur, where the center of the marker is not as easy to identify while moving. While there is no movement in a given direction, there is no motion blur, and the deviation drops.


Measurement Mean and Standard Deviation at 0.75 m/s

Measurement Mean and Standard Deviation at 1 m/s

Figure 29: Mean and one standard deviation.

The same characteristics are observed from figure 29, where the test is performed at higher velocities. (0.75 m/s and 1 m/s respectively) There is a general increase in deviation errors, and the fast-changing sections of the path are not as well represented at higher velocities.

This is somewhat expected, as the sampling rate remains constant while the velocity increases. Sharp edges will not be represented as they should, at higher velocities. This is illustrated in figure 30, where the y-component of the path is displayed. The graph representing the slowest velocity, follows the intended trajectory better than the path representing the faster tests. When examining these graphs, it is important to keep in mind that the robot will round corners, which will result in a rounded edge on the graphs, even at low velocity.

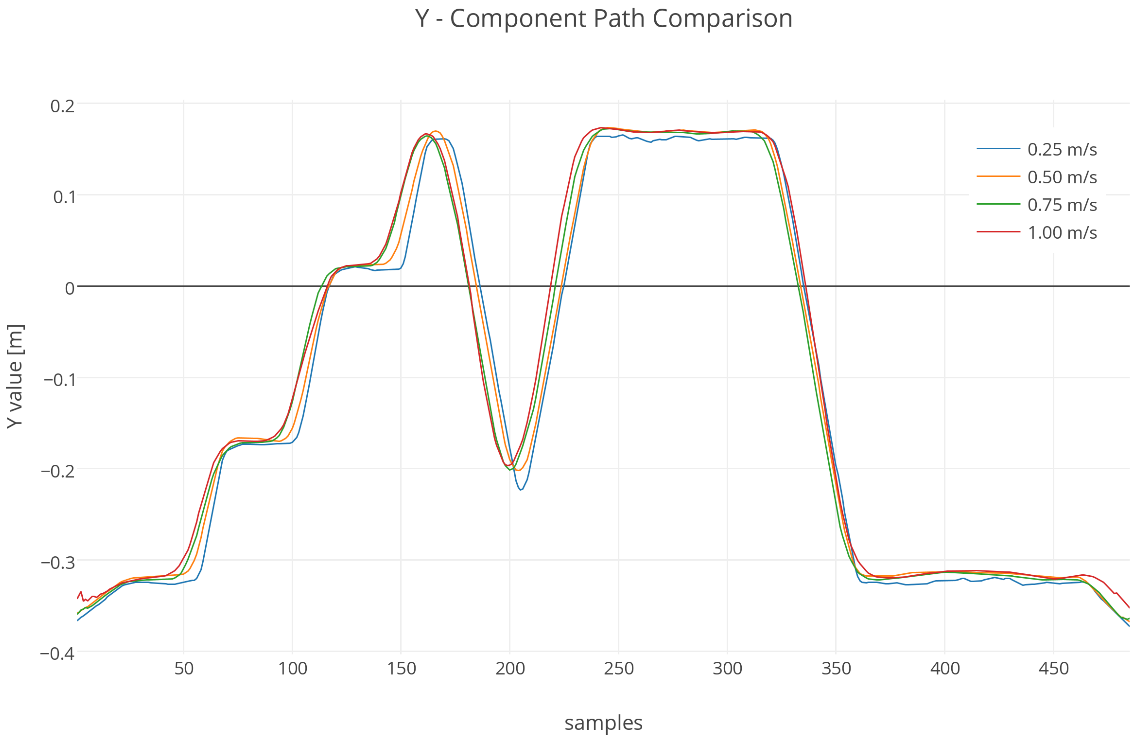


Figure 30: Comparison of paths captured at varying velocities

Lastly, an illustration of the standard deviation will be presented, in figure 31. As referenced earlier, this is an illustration of the mean standard deviation for every point in the recorded path. The standard deviation is calculated for each of the four velocities. It shows a small increase standard deviation for x and y, and a larger increase and overall value for z. The relationship between x and y is as expected, as they utilize the same measurement technique. The depth measurement uses time-of-flight technology and can be expected to show different characteristics. A contributing factor to the large deviation of z can be attributed to the shape of the marker. Had the marker been flat, a small change in x or y would not be of any significance, as the depth would be the same over a larger area. However, the marker used is spherical, meaning any deviation in x or y will also cause a deviation in z, as the surface of the sphere would be measured in different locations.

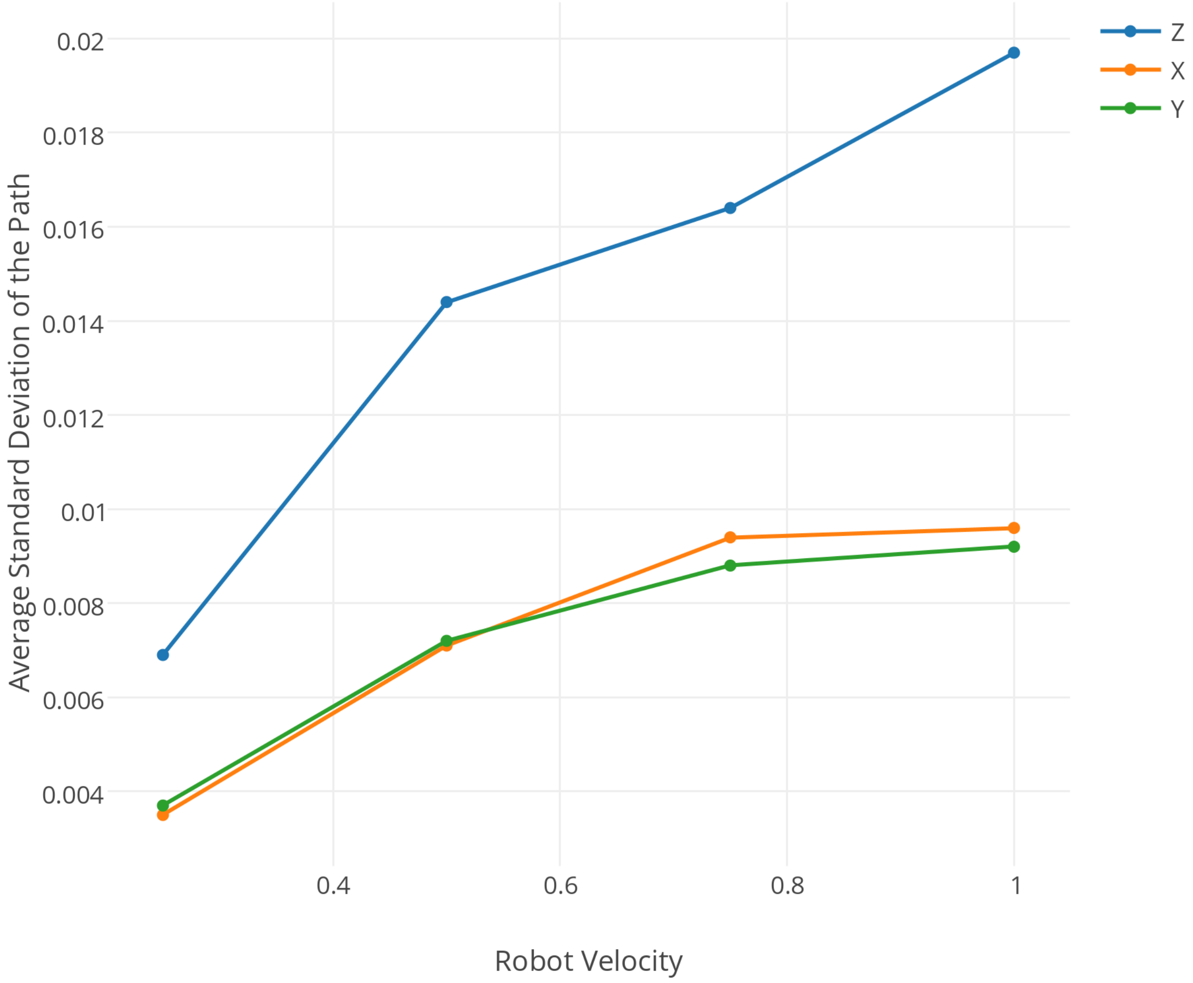


Figure 31: Average standard deviation at different velocities

To summarize, the observations made from the dynamic test at slow velocities indicated an average error of under 1 cm for both depth and height/width measurements. At higher velocities this deviation increased to approximately 2 cm and 1 cm for depth and height/with respectively. The systems ability to recreate a path proved best at slow speeds, as expected, but also showed satisfactory results, even at higher speeds. The system showed no significant indications of distortions or disturbances, except those likely caused by the marker. The results observed from these experiments show resemblance to other research done on the same subject.[19]

[19] First experiences with Kinect v2 Sensor for Close Range 3D Modelling `http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XL-5-W4/93/2015/isprsarchives-XL-5-W4-93-2015.pdf`

5 Conclusion

The measurement aspect of the Kinect device worked with high precision and accuracy. The static and dynamic tests provided satisfactory results indicating that the Kinect device is indeed well suited for precise and accurate three-dimensional measurements in an ideal environment.

However, even though the measurements made by the Kinect were good, the sensor will only provide a positional measurement (3 degrees of freedom). No orientation is measured from this implementation, which is vital to motion capture applications. Another downside is the problem of occlusion. As long as a single device is used, it will not be possible to cover the entire scene, as the user will occlude the marker.

To summarize, the Kinect provided satisfactory measurements, and fulfilled its purpose as a depth measurement sensor to a very high degree. With the addition of multiple sensors to cover a larger area of the scene and the implementation of an inertial measurement sensor, the Kinect is a very good, and well suited candidate for positional measurement in a motion capture system.

Advantages

- Provides highly accurate and precise measurements, well suited for SRP applications.
- Covers a large area of the scene, compared to the current Polhemus system.
- Unaffected by metallic objects, unlike the Polhemus system.
- Customizable by Kinect SDK, and can accept any color-based optical marker.
- Very low cost compared to current motion capture systems.

Disadvantages

- Only provides 3DOF measurements in this current implementation.
- Dependent on a suitable scene, to eliminate interfering objects.
- Problem with occlusion, as the user can block the line of sight to the marker.
- Currently no official support for cooperation between multiple sensors.

6 Improvements and Further Work

The current implementation of a motion capture system, with the Kinect as the only sensor, provided positional data only. As the marker consist of a single identifiable object (the blue ping-pong ball), any data regarding orientation is unobtainable.

There are multiple efficient solutions to this problem. The simplest might be the use of an inertial measurement unit (IMU), as an addition to the optical marker. By incorporating an IMU into the marker, it would give real-time orientational data, while the optical aspect of the motion capture system would provide positional data.

A well-suited device for this particular purpose could be the PlayStation Move device, developed by Sony for the PlayStation gaming platform. The PlayStation Move controller is a hand held device with an optical color-based marker, similar to the marker used in this implementation. In addition to possibly serving as a well-suited marker, the controller has a built-in IMU to determine the orientation. This controller could be an excellent addition to the system, as it might solve the orientation problem, as well as functioning as a commercial grade active optical marker.

Figure 32: The PlayStation Move controller. [20]

Another improvement would be the introduction of a second or third Kinect sensor. By adding more sensors, the occlusion problem would be reduced as scene coverage would increase. The Kinect sensor will currently interfere with each other, but to what degree has not yet been pinpointed. To allow for more sensors covering the same area, the sensors could be multiplexed if interference causes a problem. This feature is worked on for open source Kinect software but has no official support from Microsoft as of this moment. Promising experiments have been made by third parties.[21]

The additional sensors would have to be calibrated to use the same coordinate space, which would only require proper TR matrices to be determined from the extrinsic parameters of the setup.

[20] `http://us.playstation.com/ps3/accessories/playstation-move-motion-controller-ps3.html`
[21] `http://brekel.com/multikinectv2/`

Some small scale improvements could also be made. A growing region approach could be implemented in the thresholding process. Instead of selecting a fixed range of colors, one color could be chosen as a spawn point for the growing region algorithm. The area indicating the marker would grow, pixel by pixel from the spawn point to the point where the color changes drastically, which is the point where the marker ends. The span of color within the region could serve as a reference for determining the thresholding values.

In this calculation, the velocity of the path has not been taken into account, as only the spatial locations have been analyzed. However, each sample contains a timestamp, which will easily allow for the implementation of velocity in the captured path.

7 References

[1] Emmanouil Z. Psarakis and Georgios D. Evangelidis
An Enhanced Correlation-Based Method for Stereo Correspondence with Sub-Pixel Accuracy
`http://perception.inrialpes.fr/people/evangelidis/george_files/ICCV_2005.pdf`

[2] Hugh Liu, Grantham Pang
Accelerometer for Mobile Robot Positioning
`http://biorobotics.ri.cmu.edu/papers/sbp_papers/integrated4/liu_accel_position.pdf`

[3] Latt WT, Veluvolu KC, Ang WT
Drift-Free Position Estimation of Periodic or Quasi-Periodic Motion Using Inertial Sensors
`http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3231462/`

[4] Emmanuelle Gouillart
Scikit-image: image processing
`https://scipy-lectures.github.io/packages/scikit-image/`

[5] Pedro Burmester Campos
Adaptive Controllers using E-Puck Robots
`http://paginas.fe.up.pt/~ee06205/`

[6] *Image Recognition and Color Model*
`http://darkpgmr.tistory.com/66`

[7] Mariano Alvira, MD, LifeSpan Biosciences Inc.,
Minimizing Electronic Noise in Digital Images: example cases of controversial prostatic lesions.
`http://conganat.uninet.edu/IVCVHAP/CONFERENCIAS/Alvira/index.html`

[8] John Paul Caponigro
`http://www.johnpaulcaponigro.com/blog/tag/noise/`

[9] Johannes Kilian
Simple Image Analysis By Moments
`http://breckon.eu/toby/teaching/dip/opencv/SimpleImageAnalysisbyMoments.pdf`

[10] *3D Imaging with NI LabVIEW*
`http://www.ni.com/white-paper/14103/en/`

[11] Kinect for Xbox One
`http://commons.wikimedia.org/wiki/File:Xbox-One-Kinect.jpg`

[12] Patent: US 20080106746 A1
Depth-varying light fields for three dimensional sensing
`http://www.google.com.ar/patents/US20080106746`

[13] *Kinect for Windows features*
`https://www.microsoft.com/en-us/kinectforwindows/meetkinect/features.aspx`

[14] Victor Castaneda, Nassir Navab
Time-of-Flight and Kinect Imaging
http://campar.in.tum.de/twiki/pub/Chair/TeachingSs11Kinect/2011-DSensors_LabCourse_Kinect.pdf

[15] *Simplified Robot Programming (SRP)*
https://library.e.abb.com/public/8e8168587cb1ef4cc1257ddc0033de9f/SRP_Final_external.pdf?filename=SRP_Final_external.pdf

[16] *Simplified Robot Programming, Datasheet*
https://library.e.abb.com/public/ba0082be61601cb9c1257ddc003ad796/Simplified%20Robot%20Programming_data%20sheet.pdf?filename=Simplified%20Robot%20Programming_data%20sheet.pdf

[17] Polhemus Liberty Brochure
http://polhemus.com/_assets/img/LIBERTY_Brochure.pdf

[18] *Programming Guide: Coordinate mapping*
https://msdn.microsoft.com/en-us/library/dn785530.aspx

[19] E. Lachat, H. Macher, M.-A. Mittet, T. Landes, P. Grussenmeyer
First experiences with Kinect v2 Sensor for Close Range 3D Modelling
http://www.int-arch-photogramm-remote-sens-spatial-inf-sci.net/XL-5-W4/93/2015/isprsarchives-XL-5-W4-93-2015.pdf

[20] *PlayStationMove*
http://us.playstation.com/ps3/accessories/playstation-move-motion-controller-ps3.html

[21] *Multi Kinect v2*
http://brekel.com/multikinectv2/

[22] Tan U-X, Veluvolu KC, Latt WT, Shee CY, Riviere CN, Ang WT.
Estimating Displacement of Periodic Motion With Inertial Sensors.
http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2778319/

[23] Khoshelham K, Elberink SO.
Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications.
http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3304120/

[24] Computer Vision - A Modern Approach, 2nd Edition
By David A. Forsyth og Jean Ponce

8 Appendix

This short section will outline the miscellaneous source code for this project. There are two distinct categories of software included, the *C#* files for the Kinect software, and the Python and Matlab scripts for the processing part.

The source code files are integrated into this pdf as attachments. Two attachments are included, where each use different compression formats (zip and 7-zip) to increase compatibility.

8.1 C# Files

The application made to capture data from the Kinect is written in Visual Studio Community 2013. There are some system requirements and dependencies which have to be met, these are listed below. The source code can be downloaded from GitHub at `https://github.com/ynfin/Kinect_v1`.

- **Win 8 or Win 8.1**
  Kinect SDK requires Windows 8 / 8.1.
- **Kinect SDK 2.0**
  The Kinect SDK 2.0 libraries needs to be downloaded and installed for the application to run. This can be downloaded at `https://www.microsoft.com/en-us/download/details.aspx?id=44561`
- **EmguCV**
  EmguCV is the wrapper for OpenCV, which allows OpenCV functions to be used in a C# environment. EmguCV can be downloaded at `http://sourceforge.net/projects/emgucv/`

8.2 Python Files

The calculations and plots for the results section were done in a variety of Python scripts, each will be listed below.

- **KinectPlotter.py**
  This script will do the processing described in the results section (3.2.5). After pointing to a logfile from within the script, the file will be processed, and produce a plot file.
- **LineError.py**
  The algorithms described in 4.2.2 are done in this script. As these algorithms are replaced by higher sampling rate, and a Matlab script, this is script will only compare algorithms.

8.3 Matlab Files

Matlab was used for much of the analysis in the results section.

- **ShiftMatch.m** A Matlab script was used to produce much of the results in the results section. The algorithms for shifting the paths, and comparing them are located in this script.