Signal processing and pose estimation using accelerometers

Universiteit Utrecht

Afstudeerder:
Marcel Sondaar
0433713

Begeleiders:
dr. ir. Arjan Egges
dr. ir. A.F. van der Stappen

Abstract
When capturing the pose of actors for use in games, commercial companies often use dedicated rooms with expensive equipment to accurately establish the pose of an actor. This may be prohibitive when the location is unsuitable or the equipment too expensive. This thesis describes accelerometers as a cheap and portable sensor and it's use to perform motion capture on humans. This was done by providing a model and functions to derive the current pose based on captured data, testing its functioning under theoretical optimal conditions, as well as a small sample of live tests involving real hardware and human subjects. The tests demonstrate that pure accelerometers have intrinsic problems, and that input restrictions or alternative measuring equipment are a better path to success.

Table of Contents

Introduction

High quality motion capture has long been limited to professional companies, due to its required price of equipment, making it's application unsuited for outdoor or living room use by customers. New gaming experiences can therefore be achieved by allowing such equipment to be available in poorly controlled environments for suitable prices for consumers. This chapter describes the status quo in more detail to explain the purpose of this research.

With the introduction of the Nintendo Wii, consumers have had easy access to accelerometer hardware, it being the prominent feature of the new console. With the first release of the Wii came a boxed game called Wii Sports that demonstrated one of the practical limitations of the device, yielding games that made assumptions or otherwise puts restrictions on the input in order to classify certain motions for certain purposes. The end result was that the game used some relatively simple algorithms to translate sensor readings to game controls. A player that is aware of these methods can make use of such a method to yield alternative play styles, occasionally defeating the purpose of the exercise or making game play easier. For instance, if you play tennis in Wii Sports, the player is expected to use a full arm's swing to make the tennis racket on screen move and hit the ball. The same effect can be reached more efficiently and accurately by not swinging the arm, but only flicking the wrist, yielding in shorter movement times and invested energy while the sensor readings effectively yield the same amount of G forces, but with an improved timing accuracy to dictate the ball's direction.

Microsoft has also introduced a video solution for video-based capture as a companion to the Xbox, which relies on a pair of cameras to detect depth in the image, which is then later used to recognise body shapes and limbs. The use of cameras come with their typical downside that there are placement and occlusion issues in using the device. Full body pose estimation is typically not possible due to a restricted field of view and the presence of furniture. Occlusion and field of view issues also occur when trying to monitor multiple actors at once.

In professional motion capture, there is an implementation called Xsens which provides high quality sensors mounted on a suit to perform full-body motion capture, which was a new product on the market at the time this thesis was started and will have had its own independent research over the time that this research has been conducted. A paper discussing the prototype for force measurements dates back to 2008.[1] Unfortunately, the price of this equipment is unfortunately not within the average consumer's range.

Based on the now common introduction of accelerometers in various kinds of consumer equipment, it opens future options where motion capture can be performed with such off-the-shelf consumer hardware. This document will demonstrate some of the possibilities and problems inherent with this choice of approach.

Related work

Pose estimation using various methods have been made in the past, and some of these applications have been commercialized since. There are various applications and methodologies that are related to the presented research, and need some further explanation. First, the documented applications in the field will be discussed, followed by an overview of various types of sensors that have been used in the field and could possibly prove a viable alternative.

Pose estimation applications

While this thesis approaches the subject of pose estimation from the computer gaming and simulation industry, a lot of research has been done from other subjects of interests. The measurement and analysis of movements have found itself numerous potential and existing applications in practice. Pose estimation, when used as a form of human-computer interaction, shows that fields of physics and anatomy consistently coincide, but many other biomechanical and engineering purposes have been discussed. Since the thesis concerns basic pose estimation and the theoretical basis, the results could possibly be applied to many subject areas.

Medical sciences

In the medical world, the analysis of body movement has several healthcare purposes. Detailed observations of human pose and movements can reveal and quantify mobility. The most interesting application for this is to measure and analyse impairments in human mobility. In Schwartz et. al.[2] it is noted that observation of human movement outside dedicated recording locations is beneficial since it allows to extend observations beyond clinical visit, as well as freedom from the restraints of such observation rooms. The patient could then be observed during daily movements as well. The use of an inertial sensor would attain this goal. The specifics of this sensor are not formally discussed, although the description seems to suggest a combination of an accelerometer and a rate gyroscope. Another form of assisted medical observation that is mentioned is that recording a subject during an epileptic seizure can give insights on the responsible brain parts by matching the limbs and their quantitative activity to the corresponding brain regions.

In Zhou et. al.[3] the need for automating medical inspection is further emphasized due to a practical need caused by a lack of personnel otherwise recommended for medical treatment, pointing out stroke as a typical medical condition that is contributing to this condition. Extrapolating from this would indicate that there are numerous other medical conditions that can be assisted using computer-enabled applications. Veltink et. al.[4] also refer to this need to do home observations on subjects, without discussing the actual medical conditions or impairments of the subjects that would warrant such observation.

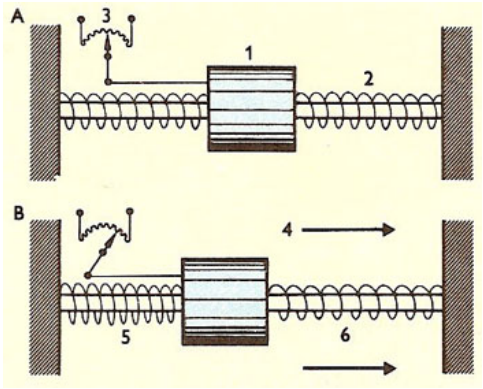
Robotics

While this thesis is in itself restricted to the human anatomy of moving limbs, there is some relevant research that deals with practical applications without necessary biological components. Stubberud et. al. briefed the NATO concerning sensors aboard unmanned vehicles.[5] While they omit details on potential military applications, the basic situation is one of keeping track of an object influenced by unknown external sources. In their topic of research, the terrain offsetting the desired route of travel for a vehicle would prove to be the challenge. The system to be discussed in this thesis also deals with unknown forces resulting in a new position of the objects under scrutiny.

Equipment

Over the course of history, many pieces of electronic equipment have been used to estimate poses. Each uses a certain aspect of physics to determine a part of the relationship of the sensor to the physical world. A wide selection of devices have selected for discussion, roughly sorted by relevance to the thesis in particular and pose estimation in general.

Accelerometer



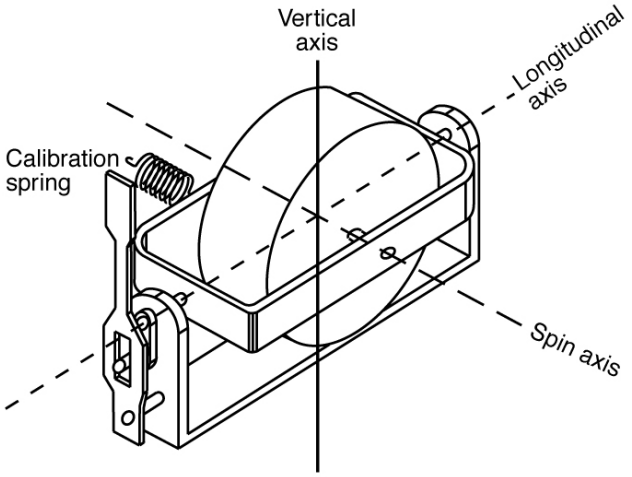
The accelerometer is a device that measures the normal force applied to the device. In its original form, the accelerometer contains a weight suspended by springs on both ends of an axis. These springs try to restore the device into its central position. When the casing of the device is moved, the inertia of weight inside tries to keep it still or moving in the same direction, and the momentum of the casing is transferred using the springs. Because a spring extends proportional to the amount of force applied, the object slightly moves from its central position proportional to the amount of force applied, and attains a new position. In an electric device, this position can be registered by a wide range of methods. The relative simplicity of the device has made it a cheap choice, and easily manufactured to smaller scales. A typical packaged accelerometer device consists of three linear units mounted orthogonally to each other to give readings covering all three dimensions of space.[6]

Since the device measures acceleration, it will always include the local gravity in its observations. This has two functional repercussions during observation. The disadvantage is that to calculate the acceleration in world coordinates requires knowing the orientation of the sensor to subtract the gravity to obtain the acceleration in other directions. The advantage of this behaviour is that if the sensor is known to be in rest, its orientation in two of the three rotation axes can be determined directly from the sensor readings without the need of knowing previous states, thereby eliminating the problem of drift.

An accelerometer alone is not enough to establish the entire state of the sensor. An accelerometer is only able to measure three degrees of freedom, whereas an arbitrary object has six degrees: three for position and three for rotation. To be able to make estimations regarding their exact pose, extra restrictions are needed. This thesis will use the physical constraints of human limbs and joints to deal with this problem. Since individual human joints have at most three degrees of freedom[7], the same as a the amount of inputs provided by the sensor, this theoretical constraint is obtained by having one sensor per joint whose configuration we want to observe.

Rate Gyroscope

An alternative device in pose estimation is the rate gyroscope. Instead of dealing with the conservation of linear momentum, it instead relies on the conservation of angular momentum. By suspending a lever from its centre of gravity and measuring the force exerted at one of the end, you will be able to record the amount of turn rather than the linear acceleration. A regular gyroscope however adds a rotating component and uses the effect of precession to reflect the force applied in one direction into a resultant force on an orthogonal axis.[8]



rate gyro

Knowing the orientation of the sensor, and especially when it's identical to the orientation of a present accelerometer allows the cancelling of the acceleration due to gravity on the accelerometer and obtain absolute accelerations and by extension absolute positioning. Since rate gyroscopes only measure rate of change rather than an absolute orientation, any implementation of absolute orientation is therefore fundamentally based on summing consecutive angular changes into the end state. On their own, these systems suffer from drift and a calibration need, which makes measurements increasingly inaccurate as time passes.

Video

The previous two methods contrast against the use of cameras to track objects. These devices consist of a large array of sensors measuring the absolute amount of incoming electromagnetic radiation over a period of time. In domestic cameras, this source is limited to visible light, but video recordings can also often be made from the infrared spectrum, which is especially used in studio-based video motion capture. Many implementations can be described as tracking marker objects over the course of time, where either a point source or specific patterns of radiations are tracked. In the case of patterns, this is either done with artificial shapes designed with the specific intent to be easily detectable[9] - such as the barcodes found on shopping articles in real life. Other implementations like SIFT[10] find and label points as a description of local details and do not depend on the actual application of dedicated markers.

The fundamental problem with this form of observation is that a camera needs line of sight to the marker to establish its relative position, after which the specification of the marker in conjunction with the camera's optical configuration will determine if it can provide additional information about it's orientation or distance. Such occlusions can be caused by environmental obstacles, or even the subjects own body – both of which might be necessary for the recording to make.

In a commercial motion capture it's typical to use point markers with a multitude of cameras. Each camera can see several markers, and by being able to draw multiple lines of sight from known camera locations, the marker would be theoretically where those lines of sight intersect. In practice, pixel sizes and other calibration and observation errors mean that the marker is to be estimated somewhere near the point where the calculated lines of sight approach each other the closest, and with a multitude of observers this estimation can be further improved.

Since this method of motion tracking has proven itself into being an industry standard, this thesis

will use this method as a reference for testing the actual performance of using accelerometers as replacement sensors.

The advantages of using local sensors compared to video capture are numerous: There is no need for a dedicated set up and configuration prior to a recording session, and there's no line of sight required meaning that the pose and location of the actor relative to its surroundings can't inadvertently obstruct observation. Furthermore, the system scales better per marker since the number of markers present means increasing markers being seen by individual cameras, which gives the problem of separating individual markers and even more complicated configurations to deal with it, while that problem does not exist when sensors report for themselves.

Magnetometer

In other research, several other devices have been considered for the purpose of pose estimation and tracking. For instance, measuring the earth's magnetic field has been an century-long standard in navigation, and in the meantime, there exist electronic versions of the old-fashioned compass – called the magnetometer. These devices have the advantage of reporting an absolute bearing, sometimes in the full 3D space rather than the typical compass' single axis of rotation. A key problem is that the while the device would normally pick up earth's magnetic field, the north-south axis can often be distorted by geological phenomenon, or locally and more profoundly, by sources of magnetic fields, rendering the device useless in the worst case.[11]

Gyrocompass

The gyrocompass is a close relative of the gyroscope, is that it uses the same physical foundations of a rate gyroscope, with the difference that it's purposely not free of resistance. Since a change in rotation applies a force in an orthogonal direction, the way of least resistance is to align with the rotational axis of the earth itself. A gyrocompass that's left in place will eventually point true north (instead of magnetic north). The downside of this device is that the time needed to establish that position is significant, and a gyrocompass is therefore incapable of accurately providing true north in an environment with continuous orientation changes beyond it's corrective ability, such as a moving human actor.[12]

Echolocation

Audio can also be used for measuring distances, and hence the location of objects. Such a system can work in one or two directions: an active beacon can send an (ultrasound) audio signal at fixed intervals, or a body or dedicated piece of electronics can respond to an audio signal. Audio doesn't travel fast compared to light and recording sensors lack directional sensitivity typical of cameras. Instead, the time it takes for a signal to arrive is used to measure the distance. With multiple sender-receiver pairs, this information can then be used to triangulate the object in question.[13]

Problem definition

In a software implementation of pose estimation, we need inputs, and an algorithm to convert these inputs to a pose. The process of determining a pose from the input of a set of accelerometers comes with a number of constraints. To be usable in practice, such an algorithm must be able to operate in real-time, and provide its answer without having access to future sensor readings. To function as a pose estimator, the algorithm to be devised must process the input sensor readings and return a pose that approximates the real-world situation.

To establish a proof of concept for this method, we limit the initial experiment to a single arm. The reason for this is that with a representative subset we can assert that qualities of the algorithm, and extend it to the entire body in a later stadium. Another reason for this choice is that in current gaming it is also the arm that is used as primary input for other sensing systems, which allows a subjective comparison. The model we use will be limited to this problem space, which includes the upper arm, lower arm, up to the palm of the hand, while assuming that the shoulder joint is fixed in space. If the player were holding an object in his hand, the joints included could make the object point in any direction, and include the majority of positions which the subject could physically reach. The fingers are assumed to hold the carried object in place, which results in the pose of the hand being fixed as needed to wield the object in question. Therefore, continuing past the wrist does not contribute much physical freedom compared to the actual degrees of freedom, and is therefore a good separation point. Adding the first few bones and joints past the shoulder add degrees of freedom to the problem, whereas they don't add as much to the volume of reachable space compared to the selected joints.

To measure this model, we would need sensors attached to each of the limbs considered movable. These would be the upper and lower arm, and the hand as a whole. The actual sensors can readily be attached to the upper arm and lower arm. Straps can be used to fix the first two sensors in place. The third sensor can be held in hand. For the sensors, Wiimotes are to be used as the readily available equipment, which contain an accelerometer within their package.
The problem is then to record the live accelerometer data, process it, and recover the original pose of the sensors. To achieve this, we will formalize the described system, and then propose an algorithm that will process this data. This thesis will finally conclude with a pair of tests of this algorithm including lab simulations and actual live recordings.

Pose definition

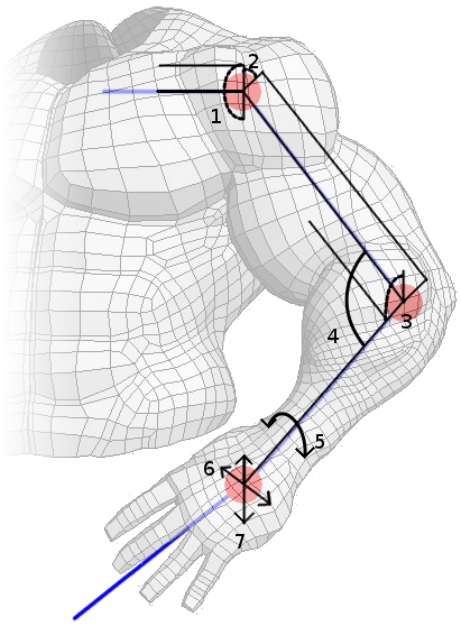
The model we will be using for the arm position consists of a skeleton with three joints and three bones. Each bone has an accelerometer attached at predefined location. The joints represent in order the shoulder joint, the elbow joint, and the wrist joint.

To simplify further calculations, we fix the shoulder joint at (0,0,0) in 3D space and define the cardinal axes of world coordinates as follows:

- The Z axis is aligned to the axis of gravity, and higher values mean larger distances to Earth's centre of mass.
- The X axis is pointing forwards from the subject, with positive values being in front of the subject and negative values being behind the subject.
- The Y axis completes the system at right angles. For a left arm which is stretched out sideways, the fingertips represent the highest Y value, and the shoulder joint the lowest value at zero. Using the opposite arm therefore lives by design in a mirrored space which

allows the same base calculations to be used after simply mirroring the appropriate axis of the sensor and the output before and after the calculation.

Overall, the model looks as follows. All the degrees of freedom are numbered according to their order:



We define J_1 as the shoulder joint, which has three degrees of freedom. These degrees of freedom are represented by three rotation matrices, so that each degree can be manipulated independently. Y-X-Y matrix rotations are used to indicate the position of the bone relative to the joint using the first two matrices, and the rotation of the bone around its axis as follows:

$$J_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(a_1) & \sin(a_1) & 0 \\ 0 & -\sin(a_1) & \cos(a_1) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(a_2) & 0 & \sin(a_2) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(a_2) & 0 & \cos(a_2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(a_3 - a_1) & \sin(a_3 - a_1) & 0 \\ 0 & -\sin(a_3 - a_1) & \cos(a_3 - a_1) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Note the subtraction in the third matrix. By including it, we essentially apply the reverse transformation provided by the first matrix, and therefore separate the rotation axis of the elbow from the direction the upper arm takes from the shoulder so that both orientations can be visualized from just their respective variable. Here, a_2 defines the angle between the upper arm bone and the Y axis matching the outstretched arm position. a_1 defines the angle around the bone where this rotation is applied. Finally, a_3 does not affect the position of the elbow joint, but only applies the rotation around the bone's axis.

The upper arm bone B_1 extends into the Y direction. A length l_1 should be provided that should approximate the typical user's distance between joints:

$$B_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & l_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Additionally, a second matrix W_1 should be added that describes the location of the accelerometer relative to this bone, usually halfway the length of the bone. We have to add at a certain offset since the volume of the sensor can't be shared with the actual body under observation, which leads to two more configurable values x_1 and y_1

$$W_1 = \begin{bmatrix} 1 & 0 & 0 & x_1 \\ 0 & 1 & 0 & \frac{l_1}{2} \\ 0 & 0 & 1 & z_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The elbow joint J_2 has one degree of freedom a_4 and operates away from the line of the previous bone, making either X or Z rotation a valid choice as the only difference is that the preceding rotation would differ by 90 degrees (observed in a_3). Z is used as the more natural choice of the pair as a quick test by bending an outstretched arm yields a rotation around world's Z axis as well:

$$J_2 = \begin{bmatrix} \cos(a_4) & \sin(a_4) & 0 & 0 \\ -\sin(a_4) & \cos(a_4) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Followed by the attached bone and sensor B_2 and W_2 according to the rules above:

$$B_2 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & l_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$W_2 = \begin{bmatrix} 1 & 0 & 0 & x_2 \\ 0 & 1 & 0 & \frac{l_2}{2} \\ 0 & 0 & 1 & z_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The wrist joint J_3 has again three degrees of freedom, of which the Y rotation is actually provided over the length of the bone and therefore provided first:

$$J_3 = \begin{bmatrix} \cos(a_5) & 0 & \sin(a_5) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(a_5) & 0 & \cos(a_5) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & \cos(a_6) & \sin(a_6) & 0 \\ 0 & -\sin(a_6) & \cos(a_6) & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(a_7) & \sin(a_7) & 0 \\ 0 & -\sin(a_7) & \cos(a_7) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

For the scope of this thesis, there is no need to define the third bone as there is no new joint to connect to, but it's definition would follow the same format as the other bones. There is however a sensor attached. In the case of a Wiimote W_3 held in the hand its orientation is no longer with it's natural Y axis aligned with the bone but rather at a right angle of it:

$$W_3 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & x_3 \\ 0 & 1 & 0 & y_3 \\ 0 & 0 & 1 & z_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

If a different grip is desired, then this can be accounted for by choosing the appropriate transformation matrices over the ones provided here.

Anticipated problems

A number of possible issues were identified before a design of algorithm was made. The rest of the chapter describes these problems and points out potential solutions or workarounds for these.

Walking noise

Using an accelerometer like the Wiimote's gives significant trouble for continuously tracking pose for a moving character. Initial tests showed that acceleration measurements when performed on a leg were repeatedly out of the 4g range provided by the input data, which also surpasses the recordable 3g range guaranteed by the device manufacturer, which simply yields values near the sensor maximum. Earlier research already demonstrated that simple walking would be sufficient to cause readings above that limit[14], even for an accelerometer that is positioned far away from the legs and consequently has the additional advantage of all intermittent joints diffusing the forces of impact from a heel strike. Since solving the problem of inaccuracies from making contact with the ground is a challenge of its own, the project scope has been moved to the upper limbs, where the range of meaningful movement is significantly larger without having to deal with overflows caused by an impact.

As a means to resolve this problem, we will have to select live experiments that don't involve walking. Instead, it is left for future research to devise implementations that would expect such behaviour. Such implementations would have to amend the algorithm with features that can handle overflows, or use specific heuristics when an (heel) impact is detected to determine the resulting pose.

Undetermined conditions

Another key issue to be observed is that there are several poses possible with the same sensor readings. This is easily shown for static poses without movement involved: Any pose rotated around the axis of gravity yields the same sensor readings as its original:

$$M \begin{bmatrix} \cos() & \sin() & 0 \\ -\sin() & \cos() & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} = M \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix}$$

This problem essentially applies to any collection of poses where the absolute orientation of the sensors can be written as another member of the set with a rotation around the axis applied.

In addition to gravity, accelerations of the body will add to the observed acceleration of the sensor, adding three unknown inputs to a system for a total of six, of which we get to observe three from the sensor alone. Corke et al. have previously stated the consequence that such ambiguity can only be resolved by using additional sensor types, or by using strong assumptions.[15] Since our system is constrained to have less degrees of freedom than the sensors provide, we can already limit the amount of ambiguity and as a consequence. Essentially the constraint of our model of the human

body forms the sufficiently strong assumption required.

Another consequence is that the algorithm has a need to store previous locations to determine the current location, demanding that that the algorithm will need to have an input corresponding to the previous configurations.

Noise accumulation

Probably the most predictable side effect of using consumer equipment is their relative lack of quality. Testing idle Wiimotes shows that the readings are not constant, even though the sensor is not displaced in any direction. Furthermore, different units have a slight difference in their 0g offset, and the transmission encoding limits the possible resolution to 0.03g. In addition, sampling happens at a fixed low rate, which means that signals with high frequency components get lost. This is especially prevalent in impact moments, where the signal spike might be smaller than the recording frequency. That net effect is that there will be an inherent drift present, and that any errors may accumulate and possibly get magnified. Any algorithm provided should therefore be able to correct for noise accumulation issues.

Algorithm implementation

Now that we have established the input constraints and theoretical requirements for the algorithm, we can proceed to defining an implementation that can satisfy the constraints mentioned in the previous chapter. The algorithm defined here can then subsequently be used in the experiments needed to assert its quality in practice.
The algorithm will consist of a fairly generic pose resolution core, and a number of formalized metrics that describe this specific application. In the rest of this chapter, each specific component will be explained in detail, followed by a discussion of the result.

The algorithm

Having established the requirements in establishing the correct pose, it becomes apparent that finding a direct equation is rather impractical. There also exists an overspecified input domain, since as input values for the algorithm we need:

1. Acceleration (a number of 3D vectors)
2. Previous pose estimation, which requires the number of degrees of freedom
3. The velocity of body parts, which can be noted as angular velocities and therefore also requires the same number of degrees of freedom.

The sum of the degrees of freedom indicates that we have thrice the number inputs for any output size, making the system overdetermined. Since the input is known to contain errors, either as input or as potentially cumulative from previous iterations the excess features in the input vector should be used to correct the movement. With the use of a fitness function the combination of constant input feature data and variable solution data can be combined to a single error score, which can then be used as the function for an error-minimising system to determine the most appropriate pose. This is a common approach to solving overspecified systems when various inputs are expected to contradict each other. An interesting application of a fitness function system has been done by Kaimakis et. al.[16] If there exists little practical knowledge of linking the input and output data sets together, trained pattern matching systems are a typical alternative choice. Since both input and output is analogue in form, using an evolutionary approach where an algorithm is generated from input data alone may provide an efficient and accurate estimation of the input data, although such methods provide little insight of why any resulting equations performs the task as expected. Zhao et. al. have applied such a method to motion capture in the past.[17]

In this case, it is easily shown how the measured data correlates with the pose and its changes. We can make motion capture recordings and obtain both the pose data and input data for reference testing. In fact, we have Newton's laws that dictate forces and movement, and we know or measure the details in the system, so we can algebraically calculate the input data given the pose and generate the measured data based on the pose.

Error minimising approach

The foundation of an error minimizer is an algorithm that can test various candidate results based on their suitability and choose a value where the error function is minimal. Since most algorithms are designed to be performed on discrete states or solutions, the choice of a solver typically requires modification to support a solution space that is linearly continuous. The search algorithm commonly

known as hill climbing is a typical candidate as we can provide an initial configuration and have it approach the optimal solution by iterating over the current solution plus or minus some value. To get an answer close to the desired solution we can choose a small difference, but as the step size decreases, the spent time will increase accordingly.

To fix this performance problem, the algorithm is run several times, halving the difference values on each iteration. This method can alternatively be viewed as a binary search for finding the solution to a continuous equation, but extended to a higher dimension.

The standard hill climbing algorithm is known for not necessarily finding the global minimum, and occasionally needs to be pushed out of that minimum and locate another one. We know however that there is a certain degree of temporal continuity in the observed states, and by using the previous solution as the starting point for finding the next solution will already provide a solution close to the global optimum. Since all the inputs being solved for are angles, we also know that each local minimum is likely repeated at a period of 2 times pi on any input from each other since these represent an identical state. These minima can however be perturbed, or completely dissolved by various other constraints. As a partial cover for these cases, the initial step size is set to 2 times pi to jump between these local minima to the most appropriate one for further refinement with smaller steps.

Static pose constraint

Since the algorithm depends on accumulating errors from constraints, we define functions that determine the values for each known criterion. This first error function is designed to provide an exact solution for the pose in a specific subset of cases, namely when there is no movement involved. This happens when the person under observation is idle.

The error function is defined as follows:

$$e(\vec{x}) = \sum w_n \left| \frac{\vec{observed}_n}{\left|\vec{observed}_n\right|} - expected_n(\vec{x}) \right|$$

Where

$\vec{observed}_n$

is the sensor reading received from from the Wiimote, and

$expected_n(\vec{x})$

is the gravitational acceleration that would be actually present if the model was in an idle state in the configuration being tested, and

w_n

is a positive number indicating the importance of the Wiimote in respect to others.

For any correct solution, the observation will be identical to the expected solution. The vector length has been introduced to make sure all error values are positive, and that small deviations in all directions are given less emphasis compared to a large deviation in a single vector component. The measurement vector has been normalized as a method to counter deviations from the expected state and measurement errors: in the idle state all Wiimotes should observe a gravity of exactly 1g. The net effect of this addition is that deviations are larger when the direction of the vector is different rather than the length, and will counter the effect of differences in Wiimote calibration.

Domain constraints

Based on initial testing it quickly became apparent that checking for locations alone would often cause the error minimising function to jump into configurations that were mirrors from the correct position, demanding an additional set of constraints for unlikely poses. One of such criteria for weeding out invalid poses is to check that they are physically possible. The joints in the body are constrained to certain angle ranges that can be measured and be included. People are still relatively different and the physical constraints for each person may be different from another (reasons include genetics, age, and training), and low ranges are used as medical indicators in various forms of physiologic treatment. Therefore we don't want to have too much discontinuity at the boundary condition to allow some movement into the inaccessible range. Therefore, we establish the error function as follows:

$$e(\vec{x}) = \sum constraint(x[n], u_n, v_n) w_n$$

with the constraint implemented as

$$constraint(x, a, b) = \begin{cases} a - x & if\ x < a \\ 0 & if\ a \geq x \geq b \\ x - b & if\ x > b \end{cases}$$

where

$$u_n, v_n$$

being the desired lower and upper bound on the input angle respectively and

$$w_n$$

being the amount of strain passing the boundary will return.

Mathematical analysis will show that anything in the expected domain yields no penalty, and that there is no value discontinuity at the boundary. This is important for the hill climbing algorithm to be guided to the correct state.

Note that the domain constraint as formulated here does not take into account any configuration that is physically impossible for other reasons than the mere limitations of joints, such as a limb sticking through the location where the subject's chest would be expected. There are several reasons why checking for this condition is undesirable:

- It requires knowledge about many more joints and physical constructs, even if they are not part of the solution to be calculated.
- It can be computationally prohibitive when done with a certain degree of accuracy.
- Subjects are not expected to hit themselves and the input which such behaviour would be designed to fix can not be expected unless the other important parts of the pose estimation fail (and thus should demand more attention).
- Disallowing such configurations may cause the pose estimator to get stuck in a configuration where any improving motion would be hindered by the presence of the hindering body, thereby preventing access to the actual optimum configuration.

Of course, a simple constraint that requires the hand to be in front of the torso may be a valid domain-specific constraint in some practical applications.

Smooth movement

Another constraint for eliminating impossible configurations is by checking for the presence of spatial coherence. By checking that momentum does not alter too rapidly in successive calculations, visual disturbances caused by input noise can be reduced or removed.

The constraints are actually a pair of error functions:

$$e_0(\vec{x}) = \sum |x[n] - p_0[n]| w_n$$

and

$$e_1(\vec{x}) = \sum |x[n] - 2\,p_0[n] + p_1[n]| w_n$$

with

p_n

Being the vector of the nth previous solution, with n = 0 for the previous configuration and n = 1 for the configuration before that. The last component being

w_n

which is used to control the influence of each component individually. The first equation gives an error boost for any change in angle, whereas the second gives an error boost for a change in angular velocity by subtracting the new angles from the extrapolated angles based on history.

At this point it is important to know that the weights should be chosen such that they do not interfere with the pose constraints ability to move the configuration. Any sufficiently high weight will simply fix the configuration in its initial configuration. Also a function controlling the second order deviation is deliberately ignored as the acceleration may change very fast in practical cases as it simply equates to applying or removing a force on the limb in question, such as releasing muscle toning causing the arm to drop under gravity. Since there is already discontinuity of force, any relevance of error functions based on the third derivative or even higher is automatically irrelevant.

The conflict between keeping momentum and position constant also have the possible positive side effect of stopping drift in cases where actual motion has stopped and the calculated system includes some leftover energy.

Dynamic pose

The all-important constraint that does the work would be the one that includes movement based on existing motion and the actual acceleration. Whereas the simple pose can cover for idle cases and low forces and therefore be possibly sufficient for observing people performing only slow movements such as Yoga and other meditative exercises, more active movement requires full use of the data available. Especially when the generated forces exceed 1g it can be mathematically demonstrated that the simple pose estimator can solve to any unrelated pose.

The core concept of the dynamic pose is that the suggested position and a few predecessors is calculated, and the acceleration present at the previous location is calculated. This effectively causes a latency of one measurement in estimating as the velocity is known between the current and previous pose, as well as the previous pose and its predecessor, which causes the actually measured acceleration to be centred on the previous sample point rather than the measured point.

The resulting equation ends up as follows:

$$e(\vec{x}) = \sum w_n \left| O_n^{-1}\left(\frac{\left(\frac{L_n(\vec{x}) - L_n(\vec{p}_0)}{\Delta t_0} - \frac{L_n(\vec{p}_0) - L_n(\vec{p}_1)}{\Delta t_1} \right)}{\frac{1}{2}\Delta t_0 + \frac{1}{2}\Delta t_1} + g \right) - \vec{observed}_n \right|$$

This construction can be divided into the calculation of velocity as the two top divisions, where

$L_n(\vec{x})$

provides the world coordinates of the nth Wiimote for the configuration, and

Δt_n

being the time difference between two subsequent observations and estimations of the sample state.

The change in velocity is then fed into the larger division which calculates the acceleration, using the average of the two time intervals to account for any changes in observation size. When acceleration caused by movement is calculated, the gravity component is added as

$$g = \begin{pmatrix} 0 \\ 0 \\ 9.81 \end{pmatrix}$$

with the value of 9.81 m/s^2 is taken from the average gravitational acceleration at sea level. The resulting vector is then transformed using

O_n^{-1}

which is the inverse of the orientation matrix of the Wiimote, with the purpose of converting world orientation to back to Wiimote relative orientation, yielding the exact sensor value the Wiimote should have after moving to this configuration, so that it can then be compared to the actual sensor reading

$\vec{observed}_n$

To complete the equation, the deviation from the expected value is again being scaled with

w_n

being the weights per Wiimote.

Combining the error functions

Since each of the functions penalizes different aspects of the configuration, certain functions will on their own converge to different solutions and need to be balanced out against each other. The general idea is to resolve the conflicts by making the constraints override each other in various cases. In general, big location deviations are significantly worse than small location deviations, so we want to prefer the location over the smoothness function when there is a relatively large amount of movement involved:

$$e_{total}(\vec{x}) = e_{smooth(0)}(\vec{x}) + e_{smooth(1)}(\vec{x}) + (e_{dynamic}(\vec{x}))^2$$

The actual function that links the candidate pose to the sensor readings is squared. This is done to make sure that if the difference is too large, that the error function will be force to solved for an accurate pose instead of dealing with edge criteria, whereas if the actual pose is close to a gimbal lock position, the pose will return low values for each of the candidate values, and the smoothness can determine which of the poses is to be preferred. Similarly, in noisy systems the actual sensor readings do not correspond to the exact poses, and a good enough pose can be used by the smoothing function to remove stuttering effects.

For the same reason, if simple fit is to be used as part of the equation it should be squared before being added to the smoothness functions.

The domain constraint has little priority demands compared to the pose estimation, as a constant error would suffice to prevent an otherwise equivalent solution of ±2π on any angle from occurring. Based on that, the hill climbing part of the algorithm will start with trying movements of exactly 2π so that it will immediately home to the most “central” minimum implied by the domain constraint without changing the actual pose.
The domain constraint does have a need to be stronger than the smoothness constraints, so that it can push the state back into the anatomically expected range the moment the pose function allows it.

In initial testing, the weights were empirically tuned to all 0.01s for smoothness constraints and all 1s for domain constraints with otherwise identical exponents to yield the intended behaviour.

Expected results

1. It is expected that this optimising algorithm will provide a significantly better answer than a predefined answer, which goes to show that the process is indeed capable of dealing with the problem. Since the fitness is an essential component of the algorithm, the failing of this hypothesis automatically invalidates the algorithm as a whole.
2. It is also expected that in cases where the speed of movement is low, the forces that alter motion are negligible compared to the actual force of gravity, and the algorithm can as a consequence be run under the assumption that there is no resultant force and calculate an answer solely based on the error estimators ignoring that fact, without it causing it to fail hypothesis 1. Testing this will show that a reduced (and thus lighter) version of the algorithm may be used in cases where violent motion is not to be expected.
3. It is also expected that under the same assumption of velocity as in hypothesis 2, the algorithm may provide an usable answer without knowing previous poses or their estimates, without failing the fitness as per hypothesis 1. In addition to the previous reasons, the independence of previous observations proves that the chosen subset of the algorithm is impervious to the consequences of accumulated errors, and as a consequence, can be run for any length of time without causing concerns for raised error margins.

Optimal settings

Due to the number of configurable settings within the algorithm, it becomes a rather mundane task of calibrating the settings by hand. Therefore it becomes another task to find the optimal configuration for the algorithm that yields the maximum performance. Any future application has its own demands, and tuning in this fashion can prove to be a good way to optimise for specific behaviour.

While there are a multitude of methods of finding an optimal solution, the choice was made for an evolutionary algorithm. This choice serves the purpose of the algorithm being potentially able to break out of local minima should the configuration need such a jump.

Expected results

1. A configuration that has been optimised for a particular setting performs better in that setting than it's initial configuration. This hypothesis serves to make sure there is improvement to be made from any human-provided setting
2. A configuration that has been optimised performs better than any other human-provided configuration setting. What we are trying to show is that a computer-optimised configuration is better than one that can be achieved by a human. While we can't possibly exhaustively test all configurations as the optimising would require the same checks, we will test this against all tests performed where the generated setting should have the best performance.
3. An optimised configuration also demonstrates improved performance compared to other ranges of tests. If an optimisation is too specific, it would not be able to adequately measure real-world scenarios where the input is of a slightly quality than assumed during testing.

Offline experiment

The experimental testing of the proposed implementation will be done in two stages. First there will be an offline test to see how well the algorithm is capable of reconstituting the system based on the input data alone. The experiment is the implementation of the algorithm and applying it to generated test data to establish the performance under the best conditions. In the second experiment actual recordings will be made with real WiiMotes compared to a baseline provided by a motion capture installation.

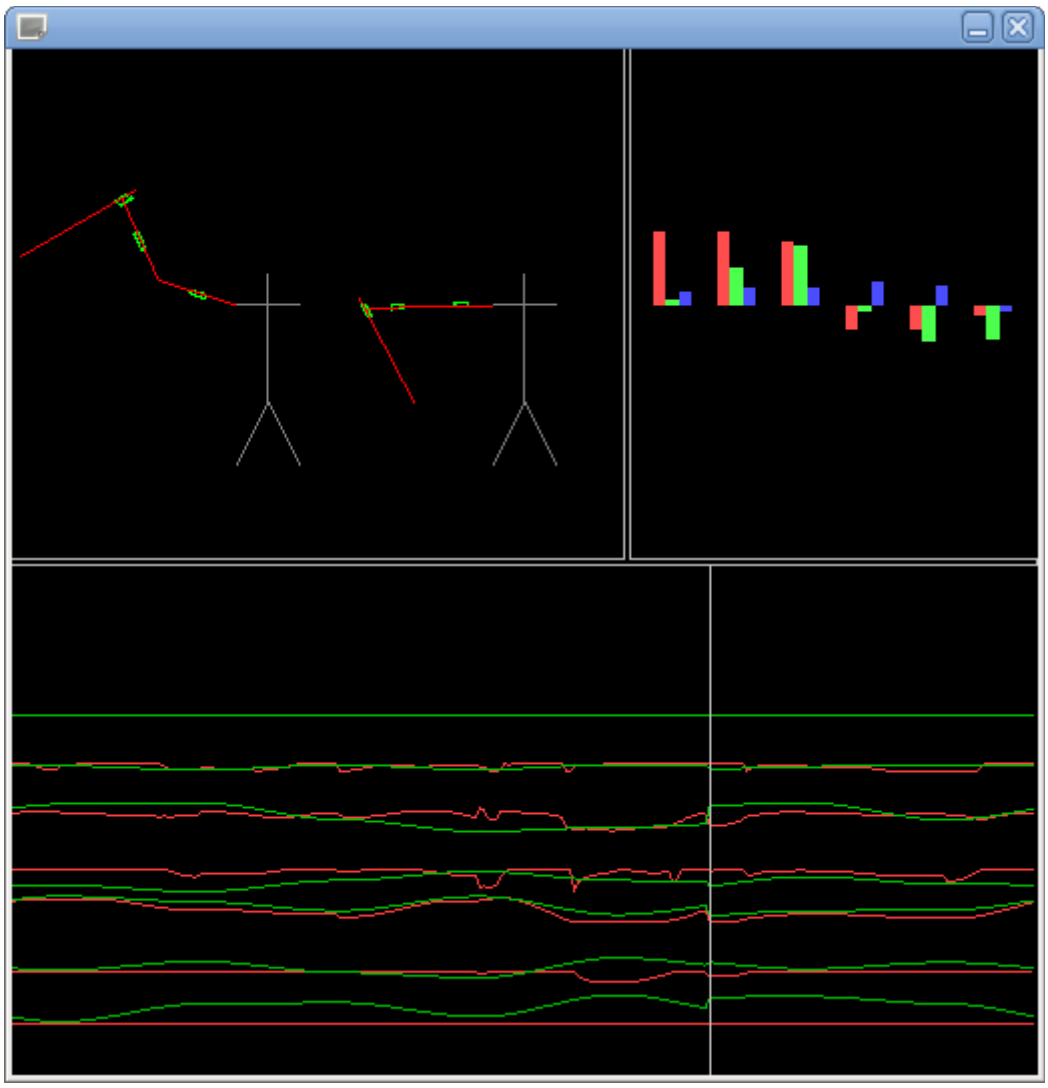
Automatic testing

The testing routine was made in two parts: one that would actually perform the test, and a pair of interfaces to the algorithm that were usable to demonstrate the live result. A visual rendition of the running test shows the data that is being observed. This output contains the three distinct parts that are key to the experiment:

- Current pose
- Current sensor readings
- Current system configuration

The pose view shows two identical skeletons, with on the left side the input data, and on the right side the estimated data from the sensor readings. It's main purpose is to quickly see if the algorithm is performing as expected and what the real-world consequences are of any deviations from the standard.
The sensor view lists the sensor readings as a bar graph with groups of three bars each representing (x,y,z). A bar of zero length represents 0g, values above represent positive g, values below represent negative g. There are two bars for each sensor: one represents the actual data, and the other represents the difference in sensor reading compared to the sensor reading corresponding to the estimated pose. Ideally, the right set of bars should be minimal in size.
The bottom half of the screen shows the configuration plot. For each variable that needs to be calculated in the system there is a corresponding pair of lines. The green line represents the input data, the red line the approximated data. There's a vertical bar across the view which indicated the current record point as the interface does not scroll the data for the ease of reading. The bars are ordered with the first variable (shoulder arguments) at the bottom and the last (wrist arguments) at the top. The distance between the red and green plot is the amount of error during calculation.

The simultaneous display of variables allows for quick observation of certain cases. If the sensor readings match the readings expected from the generated pose, while the pose shows significant differences, that often means that some constraint is missing or made an negligible impact on the problem solving. Another regularly observed comparison was that the pose matches even though the actual configuration differed. Some cases were caused by visibly identical configurations with a difference of zero modulus 2 pi, or in cases where the elbow is stretched and the wrist joint and shoulder joint have to fight over which one causes the rotation of the hand (or any of the other possible gimbal locks), or fairly often the instance occurs when the arm is in a position close to an arm outstretched to the side where the effect of the first Y rotation in the shoulder has zero effect on the actual pose and therefore the sensor readings (and typically sticks to its previous value even if the source configuration is physically altered otherwise).



Sample generation

One of the critical aspects of the simulation is that the sensor readings generated should be accurate so we do not have two concurrently running problems. Fortunately configuration settings are contrary to the input data not over-specified.

The generating process works by selecting a configuration from the sample domain as the target pose, then interpolate from the current pose to the newly chosen pose. This takes time in real life and therefore also takes time in the simulation and the algorithm will thus get a number of samples within certain intervals from one such movement. The sample rate was set to at least 30ms per frame, and less samples to allow the UI implementation to stay in real-time synchronisation. The 30 ms is based on having 30fps, which is a common maximum frame rate for a game. At that frame rate there would be 0.033s per frame, with some variation between individual frames.
The actual sample points are generated by a Hermite interpolation of the start and end point. Hermite interpolation ensures that both the angular velocity and location are continuous during the motion. The time for a full movement is chosen between 250ms and 2s, yielding both slow motions and very fast motions.

Acceleration calculations

The acceleration observed by the sensor is consistently divided into two components, the acceleration observed by the change in pose, and the acceleration observed by gravity.
The gravity component is found easily by calculating the rotation matrix for the sensor in question which transforms sensor coordinates to world coordinates. We can skip the translation matrices as they do not affect the rotation:

$$O_1 = J_1 W_1$$
$$O_2 = J_1 J_2 W_2$$
$$O_3 = J_1 J_2 J_3 W_3$$

To invert the conversion so that world coordinates translate to sensor coordinates, we need to invert the matrices, after which we can use them to convert the gravity vector:

$$\vec{a}_{g1} = (J_1 W_1)^{-1} \begin{pmatrix} 0 \\ 0 \\ g \\ 0 \end{pmatrix}$$

$$\vec{a}_{g2} = (J_1 J_2 W_2)^{-1} \begin{pmatrix} 0 \\ 0 \\ g \\ 0 \end{pmatrix}$$

$$\vec{a}_{g3} = (J_1 J_2 J_3 W_3)^{-1} \begin{pmatrix} 0 \\ 0 \\ g \\ 0 \end{pmatrix}$$

Since the inverse of a pure rotation matrix is its transposition, these equations are computationally more efficient when implemented as follows:

$$\vec{a}_{g1} = (J_1^- W_1^-)^T \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix}$$

$$\vec{a}_{g2} = (J_1^- J_2^- W_2^-)^T \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix}$$

$$\vec{a}_{g3} = (J_1^- J_2^- J_3^- W_3^-)^T \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix}$$

This requires that W is a rotation only. This can be achieved by trimming the matrix to 3x3 which drops the translation components from the matrix. The same can be done with the other rotation matrices which drops the calculation entirely into three dimensions.

The other component is acceleration based on actual movement of the sensor through world coordinates. If there exists a parametric function that describes the location of an object in 3D space, we can establish the velocity as the first order derivative of the function:

$$\vec{v} = f'(t)$$

and we can equate the acceleration as the derivative of velocity:

$$\vec{a} = f''(t)$$

The used function for pose interpolation is the Hermite function which is the third degree

polynomial:

$$h(t) = -3t^3 + 2t^2$$

Generated from the following standard properties:

$$h(0) = 0$$
$$h(1) = 1$$
$$h'(0) = 0$$
$$h'(1) = 0$$
$$1 - h(t) = h(1-t)$$

Which indicates that at t=0 and t=1 the velocity is zero, and therefore that a sequence of interpolations has a completely continuous velocity and we do not get spikes in acceleration at transitions. Since the endpoint of the interpolation is the starting point of the next, the position also has the required continuity.

The interpolation in full expands as follows:

$$a_n = s_n h(t) + e_n h(1-t)$$

With t in [0..1]. Since we interpolate over a period of time:

$$a_n(t_{current}) = s_n h(\frac{t_{current}}{t_{total}}) + e_n h(1 - \frac{t_{current}}{t_{total}})$$

Which can be rewritten as:

$$a_n(t_{current}) = s_n + (e_n - s_n) h(\frac{t_{current}}{t_{total}})$$

which provides us the angular velocity and angular acceleration as:

$$a_n'(t_{current}) = (e_n - s_n) h'(\frac{t_{current}}{t_{total}}) = (e_n - s_n)(\frac{-6 t_{current}^{\ 2}}{t_{total}^{\ 3}} + \frac{6 * t_{current}}{t_{total}^{\ 2}})$$

$$a_n''(t_{current}) = (e_n - s_n) h''(\frac{t_{current}}{t_{total}}) = (e_n - s_n)(\frac{-12 t_{current}}{t_{total}^{\ 3}} + \frac{6}{t_{total}^{\ 2}})$$

Following the above, we can establish the location of the sensors by substituting the above equation into the matrix sequences:

$$\vec{a_{m1}} = \left(J_1 W_1 \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right)''$$

$$\vec{a_{m2}} = \left(J_1 B_1 J_2 W_2 \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right)''$$

$$\vec{a_{m3}} = \left(J_1 B_1 J_2 B_2 J_3 W_3 \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right)''$$

These equations can, with a bit of handiwork, be converted into a direct equation using the standard differentiation rules and the derivatives of sine and cosine. Due to the resulting size they are omitted from this document.

The resulting acceleration can be added to the gravity component before translating it to sensor coordinates to yield the actual gravity component. Reduce the vector to three dimensions and add:

$$\vec{a}_{g1} = (J_1^- W_1^-)^T \left(\begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} + \vec{a}_{m1} \right)$$

$$\vec{a}_{g2} = (J_1^- J_2^- W_2^-)^T \left(\begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} + \vec{a}_{m2} \right)$$

$$\vec{a}_{g3} = (J_1^- J_2^- J_3^- W_3^-)^T \left(\begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} + \vec{a}_{m3} \right)$$

For an more accurate input, these samples have to be capped at 4g, aliased to the 256 sample points within that domain. Noise could eventually be added for a more realistic simulation.

Reproducing WiiMote readings

The WiiMote contains three accelerometers as part of a larger package, including Bluetooth equipment. It's programming interface is officially a trade secret, but there have since been made packages that can access the device and read data from it.[18] Since Windows, and especially its earlier versions, did only come with a small subset of supported Bluetooth device types, each client application needs to support each interface type. Windows does support the HID interface by default, so removing any Bluetooth replacement drivers gives the application an uniform access to the device under Windows.

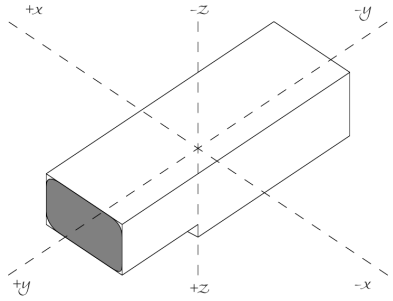
The Wiiuse package is able to connect WiiMotes and provide us with data from its instruments. The data returned from the accelerometers is sampled as an 8-bit integer number, whereas the original ADXL330 chip performs measurements on a continuous scale.[19] The original chip also mentions a minimum range of ±3g and an average range of ±3.6g in each cardinal direction.[20]

What is not specified is how the acceleration axes are mapped respective to the device, and what the relation is between the received values and the actual acceleration present. The most effective approach to determining the cardinal directions is to open the WiiMote and observe the physical orientation of the mounted chip, but that does not prevent their labelling being changed during further processing. Instead, each of the axes can be labelled by putting the WiiMote on a flat surface and printing the values for the received acceleration data. Flipping over the device so that the bottom end now faces up and taking another set of measurements will allow us to calculate the location of the up-down axis on the package relative to the device's coordinate system. The average of the two measured values will yield the reading for 0g in all directions, under the assumption that the measured values are linear respective to the input data, or otherwise mirrored around 0g.
The experiment can be repeated with each of the sides, and with the camera facing subsequently down and up.

Performing this experiment number of times yields values in the range (128±2, 128±2, 96±2) and (128±2, 128±2, 160±2), which establishes (128, 128, 128) as 0g and having 32 sample values . This also suggests that the accelerometer chip is axis-aligned to the Wiimote package, making it more intuitive for the programmer as well as making conversion of coordinates simpler. Repeating it for the other long sides yields the same difference in the first component, and either small end yields the same for the second component, reaffirming the centre of (128,128,128) and the hypothesis that

the chip is indeed aligned to the total package.

In practice, the axes of the system are oriented as the following left-handed system:



Where the numerical higher value reported by the accelerometer corresponds to a gravitational force in that direction. For example a value of z above the average of 128 corresponds to the amount of resistance provided against movement in the positive z direction.

Observing the samples shows that even when not moving, there are repeated differences between consecutive measurements. With the samples being within 2 sample points of the average, we can estimate the typical inaccuracy to be 2 / 32, or 0.06 g in either direction.

Since we know the difference between 0g and 1g readings, we can consequently test if the returned values are indeed linear for various accelerations. While it is difficult to accurately apply 2g to an axis, it is easy to apply to apply somewhere between 0 and 1 g to an axis simply by rotating the device. If the measurements are linear (to the extent the inaccuracy allows us to determine so), then the length of the observed vector relative to its centre should read 1g at all angles.
In other words, for any observation where the object is at rest the following equation should hold:

$$32 - err \leq \left\| \begin{pmatrix} x \\ y \\ z \end{pmatrix} - \begin{pmatrix} 128 \\ 128 \\ 128 \end{pmatrix} \right\| \leq 32 + err$$

This test can be performed with household equipment by positioning the Wiimote at an angle between a few books or boxes to fix it in place before reading out and recording the values from the accelerometer.

In the simulation, there is an explicit conversion to sensor readings in it's integer form, which introduces the associated measurement errors. The conversion is performed by adding 128.5 and then truncating the remainder, effectively aliasing it to the closest sensor value. The result is then truncated to the 0-255 range such that excessive g values are equally lost in the input step of the algorithm.

Finding optimal algorithm settings

As part of algorithmic testing it becomes important to know what the performance of the algorithm is so that improvements can be objectively measured. A common statistic to do this is by using the mean square error metric, which tends to emphasize the outliers, and the mean absolute error, which demonstrates the average error over the entire set.

Since the test generation knows the original configuration from which the sample data is generated, it is easy to compare the reconstructed data from the expected values. This can be used to establish the performance of the algorithm by accumulating error values over all generated data points by simply running several minutes worth of emulated poses through the algorithm to get a fair

coverage of movement options. To guarantee that algorithms each receive the same test case, the time interval will be fixed instead of operating real time like the graphical version, as well as using a seeded pseudo-random number generator to guarantee the same output in each run.

The most straightforward comparison criterion would be to compare the angles of the configuration to their expected counterpart. However as previously demonstrated, this is not a completely fair estimate as some angles can in some conditions be modified without effecting the actual pose. The other alternative is to compare sensor or joint locations for both the original configuration and the calculated configuration. This of course has the downside that orientations may be ignored while they might otherwise be a critical part of the configuration. A good metric would compare both rotation and position in a balance that is the most relevant for its specific goal.

Experimental results

Accompanying this analysis, appendix A contains a selection of detailed results of experimentally testing the algorithms with various settings. While the large volumes of test data can be compared in many ways, the graphs shown are listed according to the needs of testing the hypotheses.

The results in the first plot show a number of important base results. It contains two mostly horizontal lines: one describes an answer of only zeroes, while the other shows the free form behaviour of taking just the gravity vector into consideration. In practice, the latter tends to flip the model an 180 degrees which yields a solution that matches the input sensors, but is both physically impossible and a long distance off from the real position. Any combination of using the input function and the constraint function however shows an error margin that is decreasing when motions get successively slower, to the point where they pass under the point where an uneducated answer provides better results. However, the situation does not hold where the individual motions exceed a certain velocity. The crossing point shows to be around 500ms. Since motions are selected by two random angles within the valid domain, the average generated motion crosses a third of the domain space in that time, which extrapolates to the point where a 1.5 second time between two extremes would be where the algorithm would provide better results than an educated guess.

Plotting the statistics for the individual bones in the model it interestingly shows that the precision of the lower limbs is preferred at the cost of the upper limb, with the average answer being systematically the best in that case, whereas the improvement is much more significant in others. On the other hand, the shoulder joint provides three degrees of freedom – as much as the number of degrees provided by the accelerometer, and can often provide an entire arc of movement where the sensor readings remain the same. Therefore, using an approach that doesn't include previous states is not going to improve an answer, and explicitly finding the arc and resetting it to its centre will provide the possible optimum.

When adding any of the other constraints, it shows that the performance can go either way. Small weights provide a small improvement to the algorithm, whereas large weights tend to result in worse approximations. The effect of the additional constraints also differs with the velocity of the input sensors. The position and velocity constraints demonstrate to have a higher acceptable value for slower motions than what they do for fast motions. An additional observation is that a high position or velocity constraint produces a flat line below the uninformed answer. This is likely because the median value for each of the angles is not zero, and the function forces the answer to its most average state.

The effect of the velocity constraint appears to be the least effective, where improvements to be

gotten over the base choice happen on a case-by-case basis, suggesting it's not effective. Squaring the velocity constraint gives a more typical plot, where significant improvement can be found, with the peak settings being contested by

$$w = 0.1$$

and

$$w = 0.05$$

For the position constraint, it is shown that increased velocities renders the effect of the constraint useless. Slower motions result in an increasing number settings appearing well below the unmodified equation, while fast motions show the constraint to result in worse answers. The optimum for the slower end of the domain is near the value

$$w = 0.1$$

The squared position constraint follows essentially the same pattern, although the graph appears slightly less noisy as the base setting switches from best to worst, save from the highest weights, as well as most of the other plots reverse order of performance along with it. The optimum at lower speeds is expected to be at

$$w = 0.05$$

where lower values resolve to the original plot and the higher values tend to a frozen state.

The full pose constraint shows relatively little improvement over the base equation, and especially at the first half of the plot where it was intended to provide better performance there's no apparent improvement. An interesting observation is that the constraint values are required to be significantly smaller than those of the previous settings, with the weight for the squared constraint being approximately the square of the base constraint. This is probably due because the algorithm needs to calculate in the time of the step and therefore produces larger error margins for the poor input quality compared to the time frame.

Live experiment

Since any implementation will be run on actual subjects, it would be prudent that a test is performed with real-world data as opposed to generated data. For this, the subject will be put in a motion capture suit, and the WiiMotes will be strapped on top of the suit. The motion capture system used provides accuracy in the order of millimeters. The previous experiment shows that the observations are on average off by at least 10 centimeters. This factor 100 difference in accuracy makes the reference motion capture system an effective ground truth.

Inspired by the Wii's original purpose of playing games, a set of nine motions were chosen. These would have to be either mostly slow (favourable to the algorithm), fast (not favourable), and things in between. Individual items are selected to cover a broad and representative spectrum of possible motions. The list includes the sports included in the game of Wii Sports, the game that's bundled along with a new machine. The entire list consists of:

- Boxing. Fast jabbing motions between bounces in the upper arms. A regular boxer will never stand still, giving the algorithm continuous amounts of accelerations in all directions to deal with.
- Sword fight. The medieval style involves swinging a fairly heavy piece of metal, which means long curves and lots of changes in directions, and their corresponding G forces. Games like Zelda: Twilight Princess expects the user to make very broad swings to trigger special abilities.
- Fencing. A different style, where there's very little movement in the upper arm compared to the wrist, which has to quickly manoeuvre in all directions to keep the opponent's rapier at a distance. This will most likely stress test the third joint.
- Tennis. Unlike boxing, there's a time of rest between individual swings, making this a lighter test.
- Golf. Like tennis, this is also part of the original Wii Sports games. The user is expected to use the entire arm length to make one large swing, with times of rest in between.
- Billiards. Also consists of a sharp moment between moments of rest.
- Bowling. This is more a sport of control and starts the easy list. There's one swing, but there is no need for particular accelerations as you would expect in a golf swing, although there are significant differences in placing
- Shooting range. In motion, very little will happen, apart from aiming and potentially the acted feedback from the gun.
- Driving. The steering motion tilts the arms into different positions normally indicating the direction of the steering wheel. An educated driver can be expected to remain calm, giving smooth curves and little acceleration to deal with.

The test subjects will be given just the name of the motion, and the instruction to perform this in the way of their own making, deliberately leaving them unaware about the purpose of each recording.

The recordings were performed with two participants, both were healthy males between 20 and 30 years of age.

Post-processing

The motion capture data has to be processed, and the angles of the joints according to the proposed system have to be calculated, in order to establish the ground truth. Technical constraints prevent the motion capture software to record synchronously with the bluetooth connection reading the WiiMotes, which is read by a separate program. To realign the recordings, the angles are converted back to WiiMote acceleration data using the same logic as in the first experiment. This gives two plots for each input variable, which can then be shifted and manually overlaid to find their relative shift. To test for the possibility of clock drift, the registration is performed using either end of the plot and then compared if there's any change between measurements. After registration, the plots are trimmed, and the selection of WiiMote data is passed through the algorithm to create joint angles. These are then compared against the ground truth.

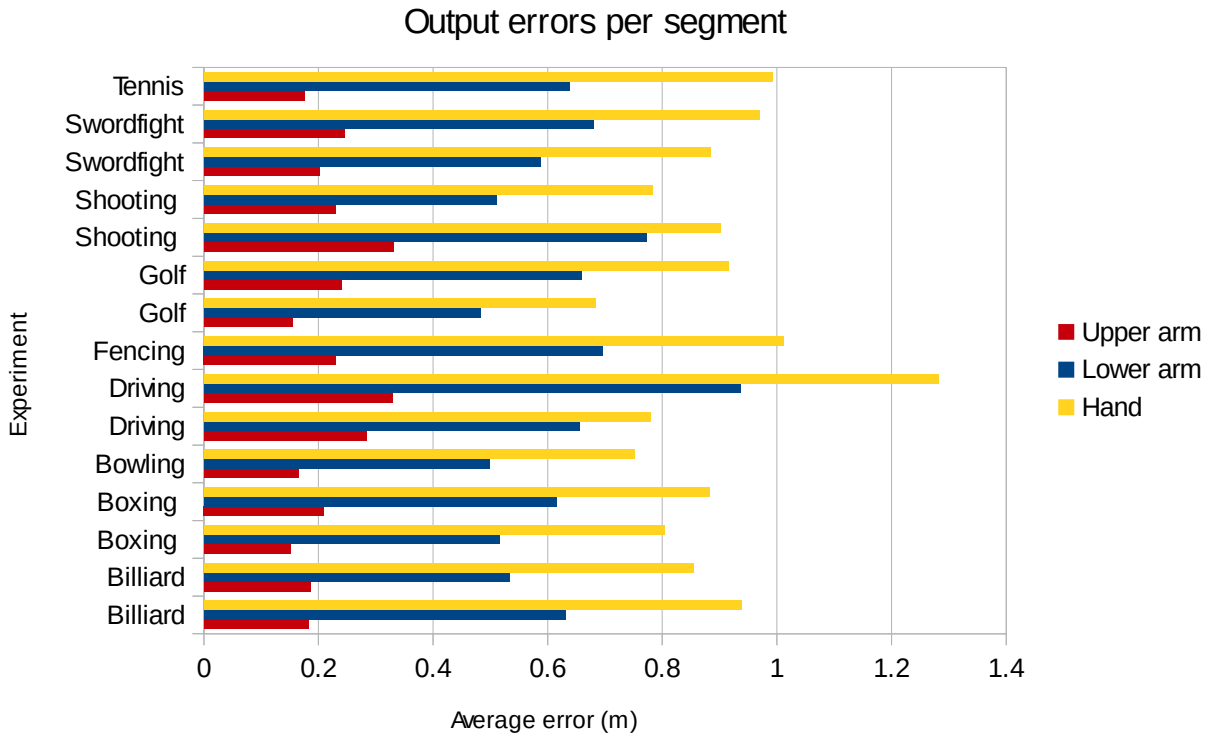
These recordings immediately demonstrate that it's problematic to actually keep the sensors located in place with the correct orientation. This manifests itself in plots where parts of the graph reappear partially or predominantly on different axes compared to their originals. In addition, the G-sensor recordings show their intrinsic nose, as well as noise caused by the sensor seemingly shifting in position during the recording. A few recordings were impossible to register as a consequence.

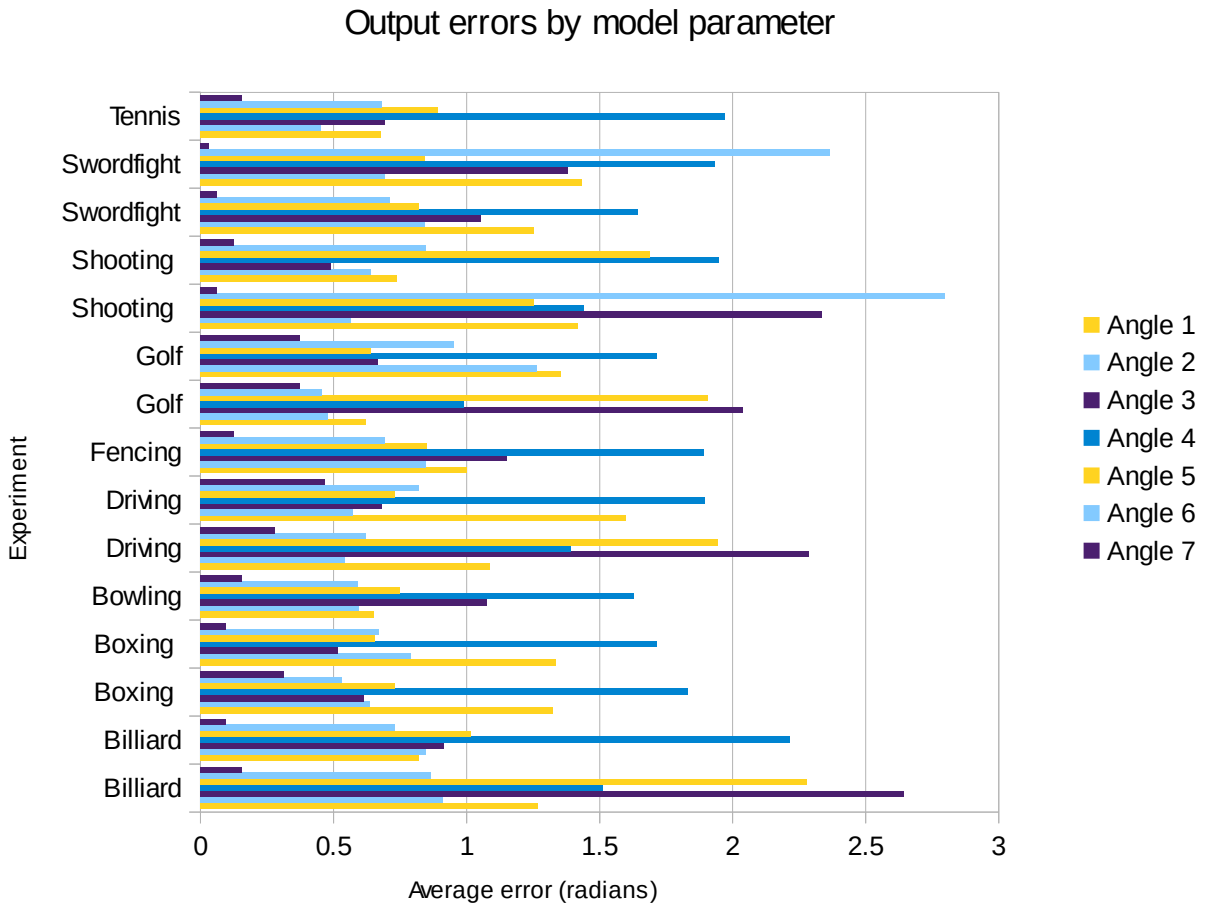
To get a quantified quality of each recording, the reconstructed angles from the wiimotes are compared on a per-sample basis to the ground truth, after which error metrics can be established. Possible error metrics would be the mean (square) error between each of the angles, and the mean error between the actual locations in space. The former has the problem that differences in angles can give arbitrary values when the axis in question is near a state of gimbal lock, where large differences on one axis have little effect, or where one axis aliases for another and both can move freely as long as their sum remain the same. Potential problems with the latter is that the location is problematic to define, and that errors in earlier joints have a potentially magnifying effect on the location error at the end of the system. To provide an meaningful solution to this, the locations are taken at the end of each limb. This location is apparent for the first two limbs, but for the last limb this would mean the size of the person's hand, which is small compared to the rest of the system. Instead we assume the person holds a virtual object as in an actual game, with the WiiMote being the handhold, and the end of the limb being at the tip of the object, 50 cm away.

Experimental results

The experiment results in a series of individual pairs of angle vectors consisting of a vector for each frame of input data after running the algorithm, and one for the reference as derived from motion capture. Since the purpose is to determine the quality, the actual vectors and individual frames have little meaning, and instead the difference between each pair of vectors is calculated and accumulated into error distribution plots. The analysis covers many details from individual experiments, which have been aggregated into the following collective results, that give a good indication of the average performance of the algorithm.

The following graphs shows the average distance between the end of each limb and it's actual position, and the average difference between each solved angle a_n and its ground truth:





Further test results can be found in Appendix B, which shows the individual error distributions as histogram plots. The relevant details thereof are briefly mentioned in the analysis.

This second experiment illustrates the problems that occur when using this method in practice. Observations show that in every experiment, the calculated hand position is half a meter away from the actual location on average, which is a significant error that renders the current implementation insufficient. This average distance is as unperformant and often worse as the average distance that would have been achieved by supplying an answer consisting of all zeros.

If we look at the detail plots, we see that the first two angles – the ones governing the upper arm take the form of a gaussian curve, and especially so in the second angle which determines the deviation of it's horizontal position. The plots also show that the mean is not centred around a difference of 0 radians, but rather off-center, indicating a bias. Since the actual positions of the devices aren't calibrated in any way, adding a system that can cover for the error between the actual placement and the intended placement of the sensor could possibly improve the final output.

One notable outlier in the upper arm plots is the graph corresponding to the shooting task. During the recording, the participant held his arm stretched in horizontal position, with occasional emulation of knockback from the imaginary gun the subject was holding. This essentially means that the first and second sensor were consistently placed in-line with each other and the system was not physically given an opportunity to establish the actual orientation of the elbow's joint rotation axis. In mathematical terms, this rotation axis is the cross-product of the directions of the upper and lower arm. With the cross-product defined as $a \times b = \|a\| \cdot \|b\| \cdot \sin\theta \cdot \vec{n}$ and the theta corresponding to the angle between the limbs being continuously close to zero, we can see that the result of the product is also a vector of nearly zero length, effectively losing against noise influences.

The results for the elbow joint itself poses more serious problems, as the distribution plot seems to indicate that the computed angle is never accurate in respect to the actually reported angle, leaving a blank space around the 0 radians of error. Instead the plots shows a peak at $\pi/2$ radians, or a quarter circle deviation, or have seemed to mirror the angle altogether. The effect of such a misplacement obviously has unfortunate consequences on the final output.

A similar pattern appears in the wrist joints, which both seem to be the sum of two gaussians, their peaks at similar distances from the centre. This similarity implies that this joint belongs to the same solving issue. Further investigation will have to reveal what the source of this problem is, and how it relates to the current implementation.

Conclusion

In this thesis we have investigated the application of accelerometers available as off-the-shelf consumer hardware to perform motion capture on subjects. To this end we have performed two experiments, one to test our approach in theoretical terms, and one to test the applicability of this method in practice. While we have established that the pose can be approximated within a certain norm, the problem is not solved in the case where fast motions are required, which limits the practical use of the algorithm to only a subset of use cases. For those other cases where the tracking of fast movement is required, a different solution or algorithm is needed. There have been further difficulties with applying the method to actual hardware and subjects, and a lot of work has to be done in that area.

One of the reasons for this theoretical limit is that the full pose constraint as suggested has limited effect on the final result. For practical use of the full pose constraint, it may be wise to multiply the constraint formula by the time per frame to maintain better maintenance of settings when varying conditions.

One of the key observations of the second experiment is that the differences between the simulated environment and an actual implementation is that the differences are very large, with nearly twice the error as the optimum expected from the first experiment for the second and last bone. This has to be caused by the system which can't properly cope with errors introduced by a live environment. The results are still potentially usable when they are just limited to the upper limb, but for lower positions in the model, the accuracy has demonstrated to be insufficient, and further work needs to be done so that the effect of these errors can potentially be reduced.

One of the other contributing observations is that the orientation of the sensors seem to change over the course of the experiment, making it necessary to reposition sensors, or accounting for change during progress. The bulkiness of the used sensors can have had a big contributing factor to input errors, since smaller sensors need less force to be moved in conjunction with the subject.

Future work

Further developments might include other forms of post-processing. In several cases, it is possible to detect that the observed vector has significant portions that do not correspond to the gravity vector. Many other sources in the field have suggested the use of a Kalman filter to determine a final result based on a combination of input and expected future state.[21][22] To use this algorithm, the statistical deviations need to be known at computation time. While the deviation of the individual components is known by its technical specification, the deviation of the computed pose is not known. Further tests can establish the needed deviations needed to implement the filter.

In the meantime, an increasing subset of consumer-enabled hardware contains the complementary pair of inertial sensors, with the accelerometer being one, and a solid-state gyroscope being the other. The combination has been demonstrated to be sufficient to compute individual directions by simple integration from the gyroscope only. This significantly improves the performance of the process as the pose can be directly established from the gyroscope orientation.

Within the process as described by this thesis, potential improvement can be achieved by dealing with input noise. The static noise given by a sensor that's perfectly still can to some extent be combated by using a sensor with a higher mechanical accuracy, or by specific calculations that compensate for the errors caused by friction and mechanical latencies generated by the internal movement of the sensor.

Proper mounting of sensors, as well as calibration of the actual model to the actor in question can also help deal with the effect of sensors being placed on the subject in locations that deviate from what the algorithm was originally expecting. Limb lengths vary from subject to subject and have a noticeable effect on the forces that are observed by the sensor as per the following common equation for centripetal force:

$$F = m r \omega$$

This shows that the size of the limb is directly proportional to the force generated by moving it.

More research can be performed in determining what sources of noise have what kind of impact on the final outcome, both quantitative, and in ways how to filter or calibrate for specific sources.

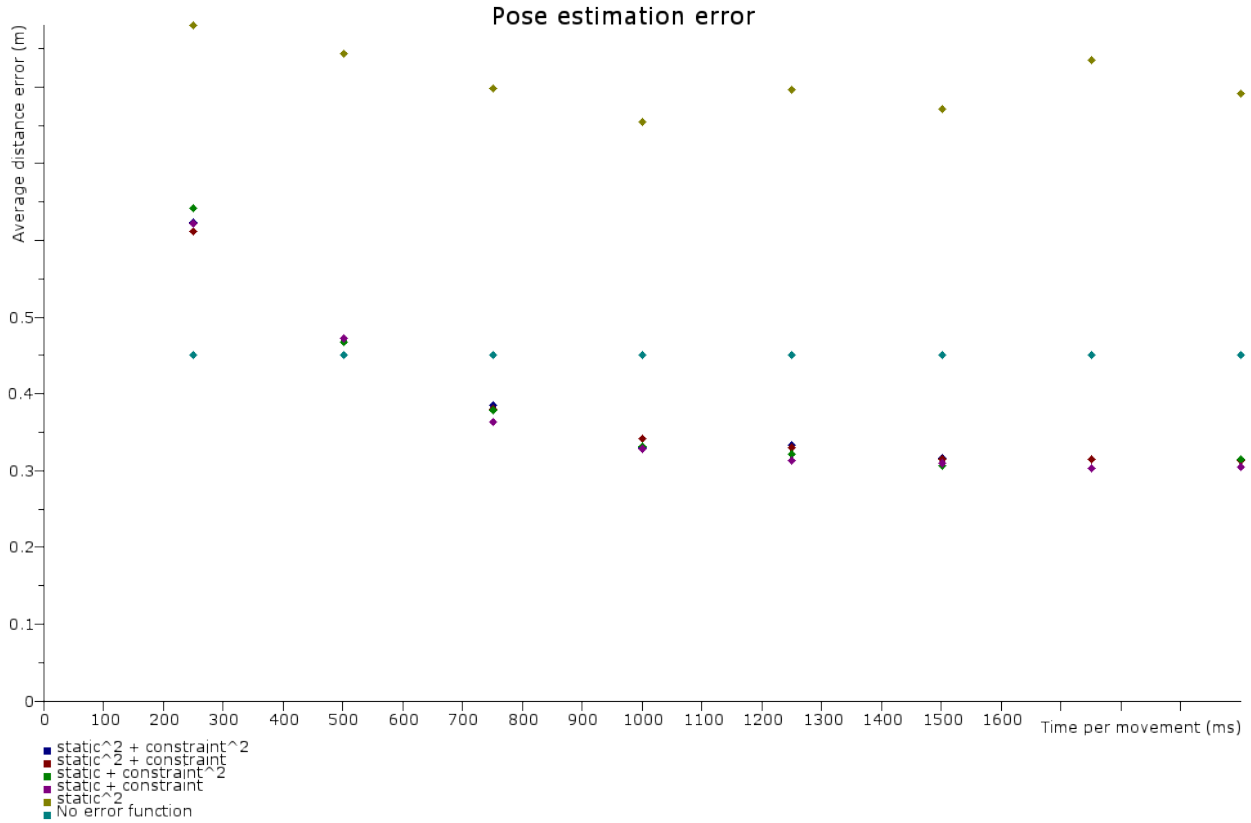
In continuation of this work, the next key step is to find a way that avoids the mirroring issues found in the current implementation, as they are the current biggest cause of output error in its current state.

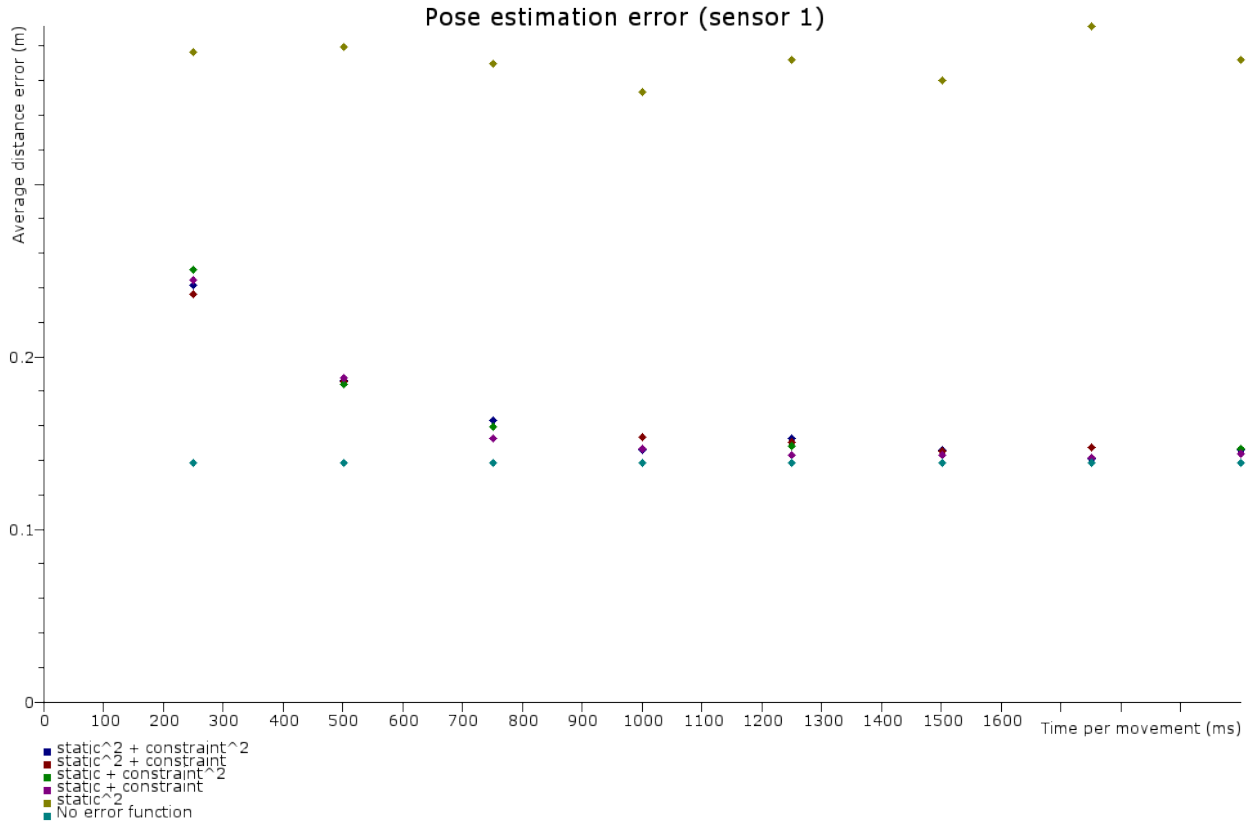
Bibliography

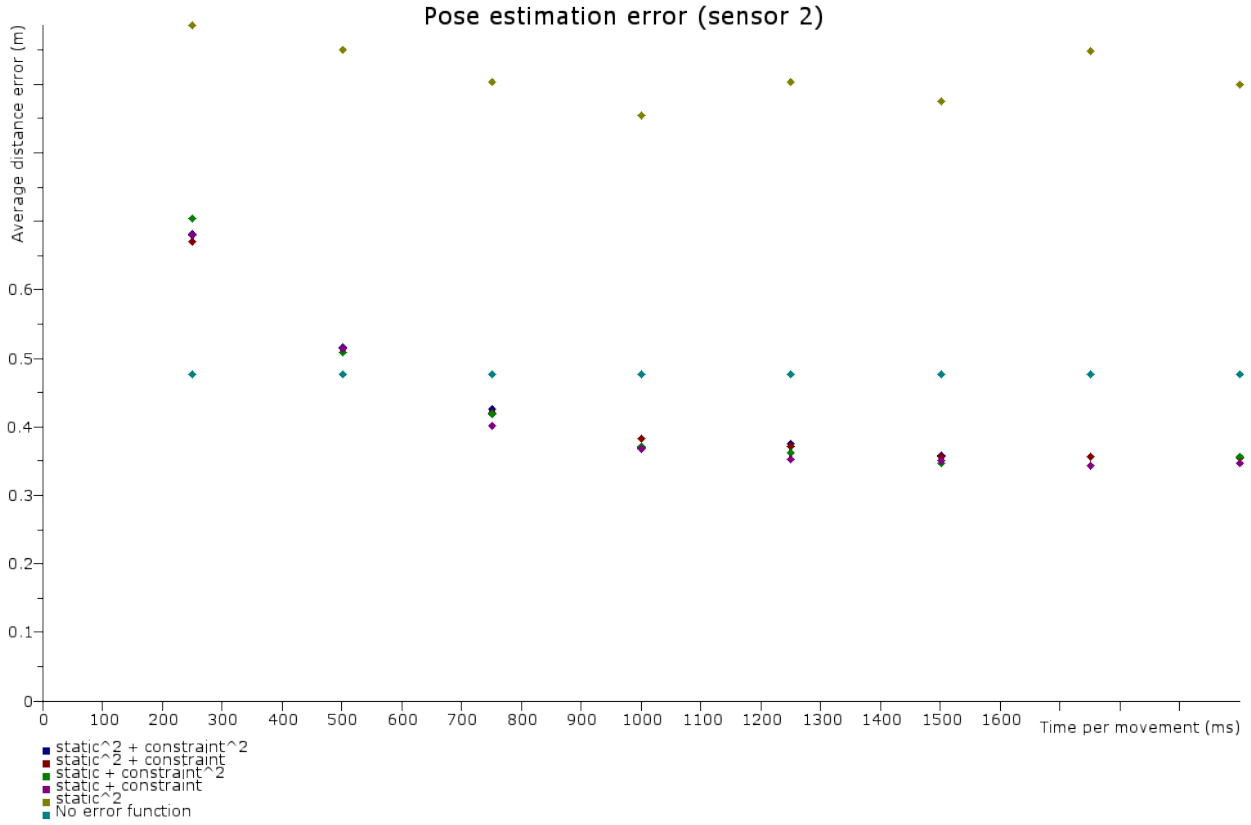
1. Xsens 3D Motion Tracking. “Research” http://www.xsens.com/research (retrieved 06-11-2011)
2. Schwarz, L.A., Mateus, D., and Navab, N. "Discriminative human full-body pose estimation from wearable inertial sensor data." In *Modelling the Physiological Human*, pp. 159-172. Springer Berlin Heidelberg, 2009.
3. Zhou, H., et al. "Use of multiple wearable inertial sensors in upper limb motion tracking." *Medical Engineering & Physics* 30.1 (2008): 123-133.
4. Veltink, P. H., et al. "Detection of static and dynamic activities using uniaxial accelerometers." *Rehabilitation Engineering, IEEE Transactions on* 4.4 (1996): 375-385.
5. Stubberud, A. R., and Xiao-Hua Yu. *Signal Processing for Micro Inertial Sensors*. Department of Electrical and Computer Engineering , University of California, Irvine, 2000.
6. Darling, D. “The encyclopedia of Science.” http://www.daviddarling.info/encyclopedia/A/accelerometer.html (retrieved 07-02-2015)
7. Urtasun, R., Slides for the 2010 course of Human motion analysis, University of Toronto, Department of Computer Science. http://www.cs.toronto.edu/~urtasun/courses/ETH10/human_motion_analysis.html (retrieved 21-01-2015)
8. Aviation supplies and Academics. “Aviation dictionary.” http://www.datwiki.net/page.php?id=6532 (retrieved 07-02-2015)
9. Kato, H. et al. “ARToolkit”, University of Washington. http://www.hitl.washington.edu/artoolkit/ (retrieved 08-02-2015)
10. Lowe, R. “Distinctive Image Features from Scale-Invariant Keypoint.” University of British Columbia, 2004
11. San Francisco National Maritime Park Association. “Submarine electrical systems.” http://maritime.org/doc/fleetsub/elect/chap17.htm (retrieved 07-02-2015)
12. Wikipedia. “Magnetometer.” http://en.wikipedia.org/wiki/Magnetometer (retrieved 07-02-2015)
13. Wikipedia “Acoustic location.” http://en.wikipedia.org/wiki/Acoustic_location (retrieved 07-02-2015)
14. Light, L. H. et al. “Skeletal transients on heel strike in normal walking with different footwear.” Journal of Biomechanics Volume 13, Issue 6
15. Corke, P., Jorge, L., and Jorge, D. "An introduction to inertial and visual sensing." *The International Journal of Robotics Research* 26.6 (2007): 519-535.
16. Kaimakis, P., and Lasenby, J. "Markerless motion capture with single and multiple cameras." In *Image Processing, 2004. ICIP'04. 2004 International Conference on*, vol. 4, pp. 2607-2610. IEEE, 2004.
17. Zhao, Xu, and Yuncai Liu. "Generative tracking of 3D human motion by hierarchical annealed genetic algorithm." *Pattern Recognition* 41, no. 8 (2008): 2470-2483.
18. Sourceforge. “Wiiuse” http://sourceforge.net/projects/wiiuse/ (retrieved 20-04-2013)

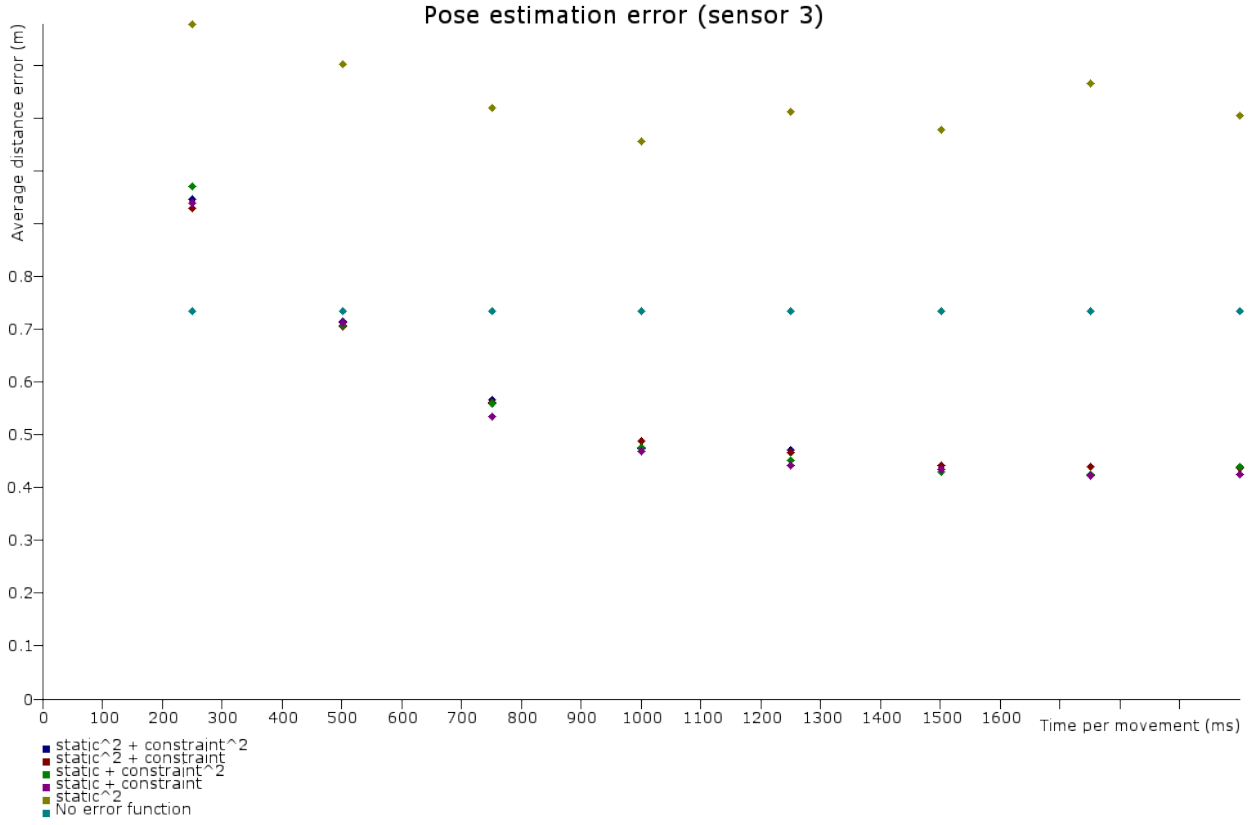
19. Analog Devices. “Analog Devices and Nintendo collaboration drives video game innovation with iMEMS motion signal processing technology.”, May 2006. http://www.analog.com/en/press-release/May_09_2006_ADI_Nintendo_Collaboration/press.html (retrieved 08-02-2015)
20. Analog Devices. “ADXL330 - Small, Low Power, 3-Axis ±3 g iMEMS Accelerometer Data Sheet (Rev. A)”, 2007.
21. Roumeliotis, S. I., Johnson, A. E., and Montgomery, J. F. "Augmenting inertial navigation with image-based motion estimation." *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*. Vol. 4. IEEE, 2002.
22. Vlasic, D. et al. 2007. “Practical Motion Capture in Everyday Surroundings.” ACM Trans. Graph. 26, 3, Article 35 (July 2007), 9 pages. DOI = 10.1145/1239451.1239486 http://doi.acm.org/10.1145/1239451.1239486.
23. Schall, G., et al. "Global pose estimation using multi-sensor fusion for outdoor augmented reality." *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*. IEEE, 2009.
24. Rehbinder, H., and Ghosh, B. K. "Pose estimation using line-based dynamic vision and inertial sensors." *Automatic Control, IEEE Transactions on* 48.2 (2003): 186-199.
25. Foxlin, E., and Harrington, M. "Weartrack: A self-referenced head and hand tracker for wearable computers and portable vr." *Wearable Computers, The Fourth International Symposium on*. IEEE, 2000.
26. Baillot, Y., et al. "Evaluation of the ShapeTape tracker for wearable, mobile interaction." *Virtual Reality, 2003. Proceedings. IEEE*. IEEE, 2003.
27. Grejner-Brzezinska, D. A., et al. "Multi-sensor personal navigator supported by human motion dynamics model." *3rd IAG/12th FIG Symposium*. 2006.
28. Feiner, S., et al. "A touring machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment." *Personal Technologies* 1.4 (1997): 208-217.
29. Kelly, J., and Sukhatme, G. S. "Visual-inertial sensor fusion: Localization, mapping and sensor-to-sensor self-calibration." *The International Journal of Robotics Research* 30.1 (2011): 56-79.
30. Gordon, Gaile, et al. "The use of dense stereo range data in augmented reality." *Proceedings of the 1st International Symposium on Mixed and Augmented Reality*. IEEE Computer Society, 2002.

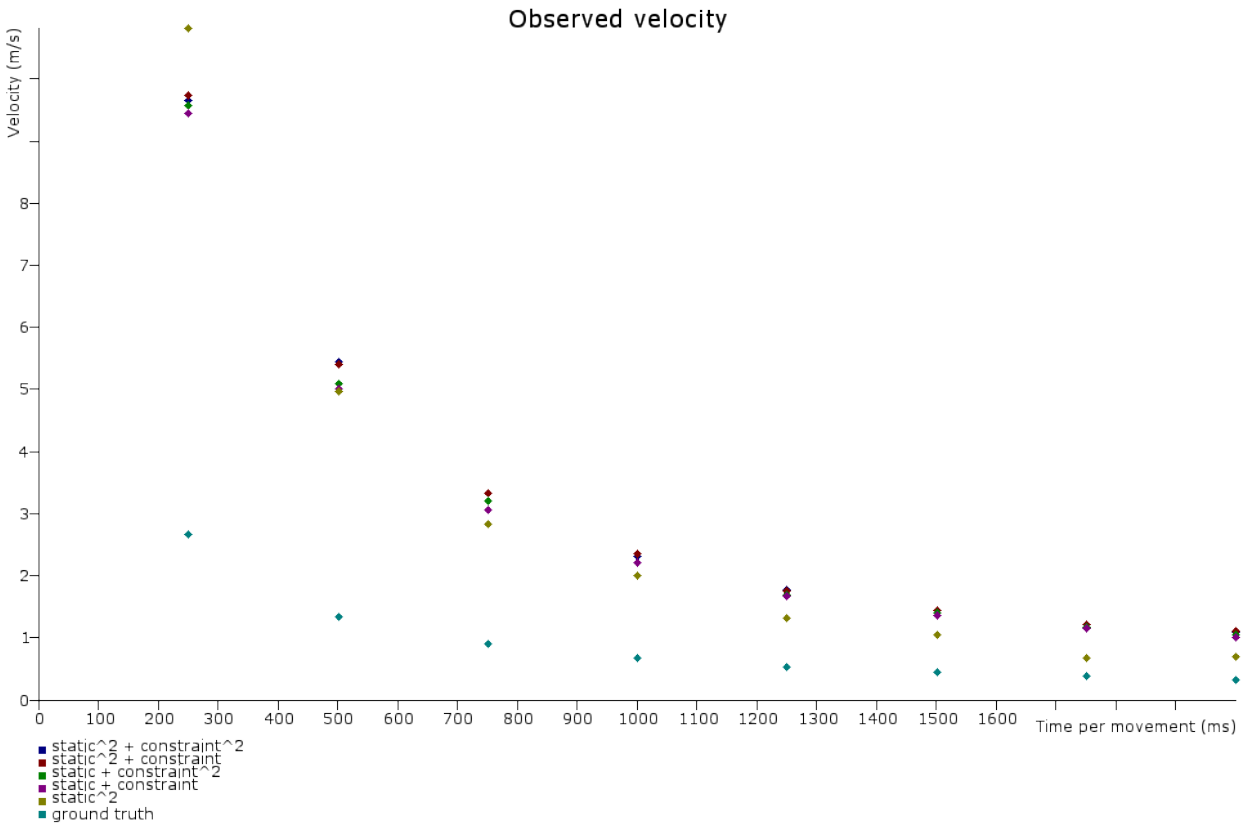
Appendix A – Raw results from the simulations

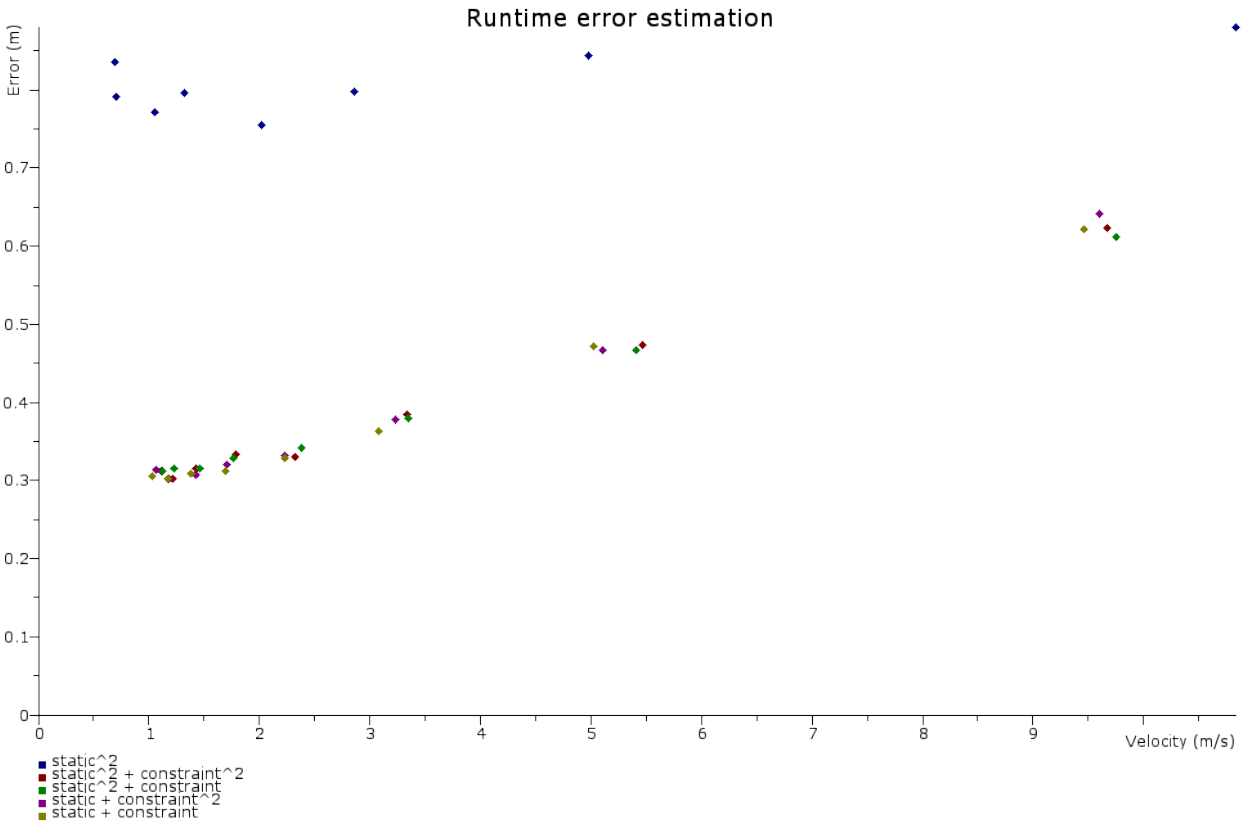




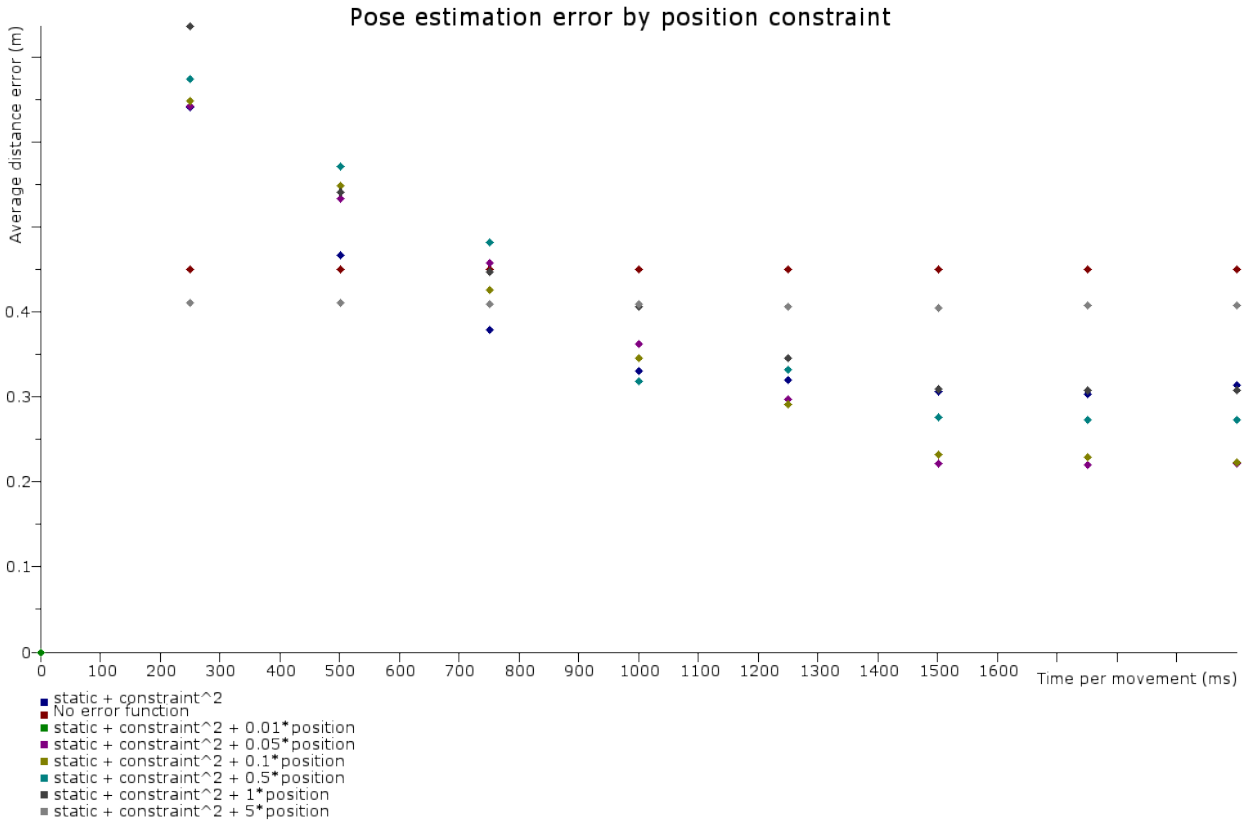

Pose estimation error (sensor 2)
Average distance error (m)
0
0.1
0.2
0.3
0.4
0.5
0.6
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
Time per movement (ms)
static^2 + constraint^2
static^2 + constraint
static + constraint^2
static + constraint
static^2
No error function

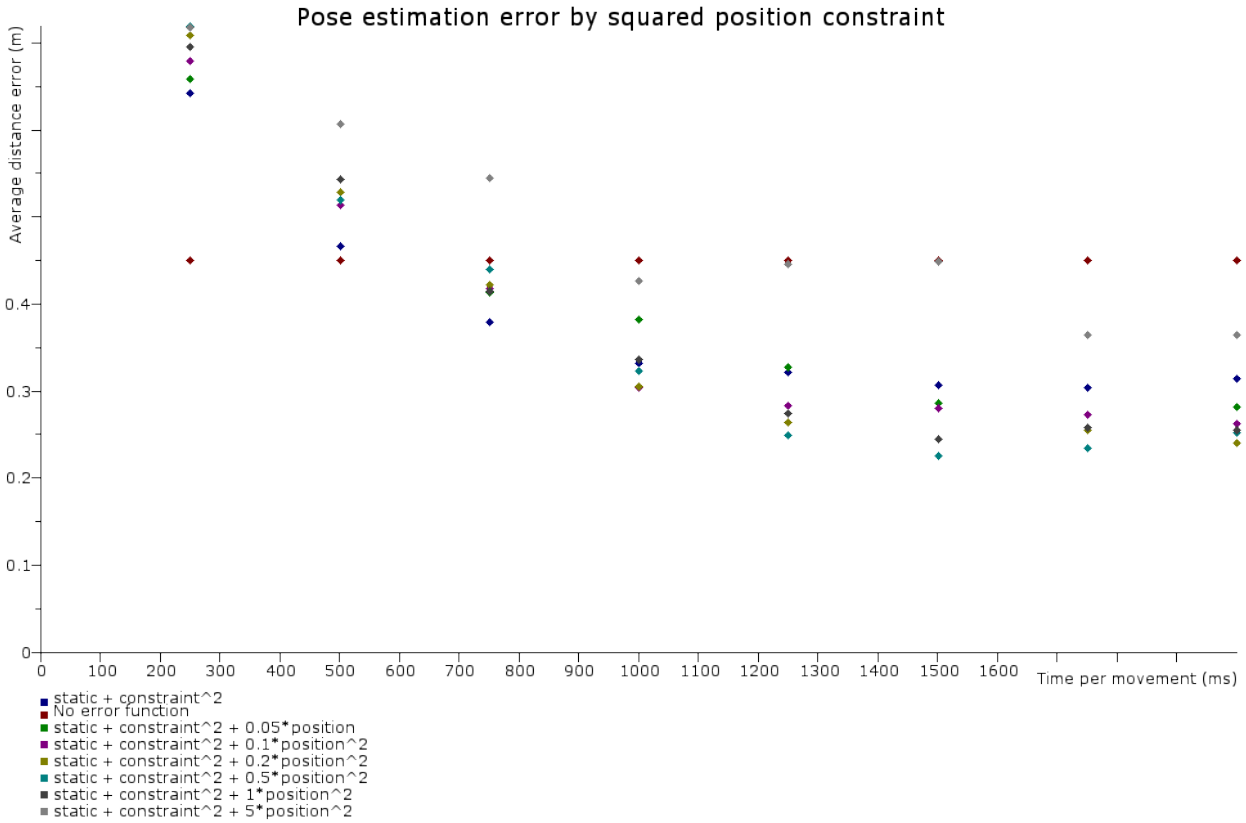

Pose estimation error (sensor 3)
Average distance error (m)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
Time per movement (ms)
static^2 + constraint^2
static^2 + constraint
static + constraint^2
static + constraint
static^2
No error function

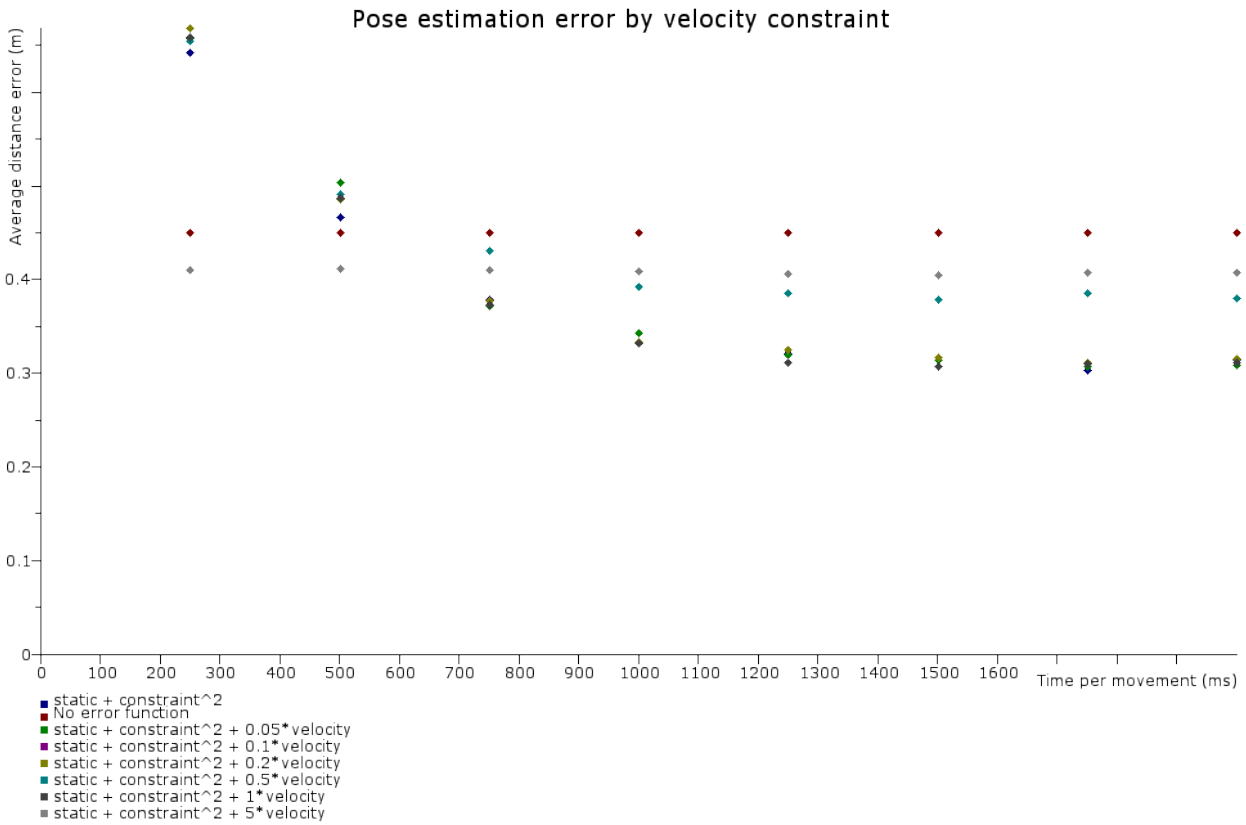

Observed velocity
Velocity (m/s)
0
1
2
3
4
5
6
7
8
0
100
200
300
400
500
600
700
800
900
1000
1100
1200
1300
1400
1500
1600
Time per movement (ms)
static^2 + constraint^2
static^2 + constraint
static + constraint^2
static + constraint
static^2
ground truth

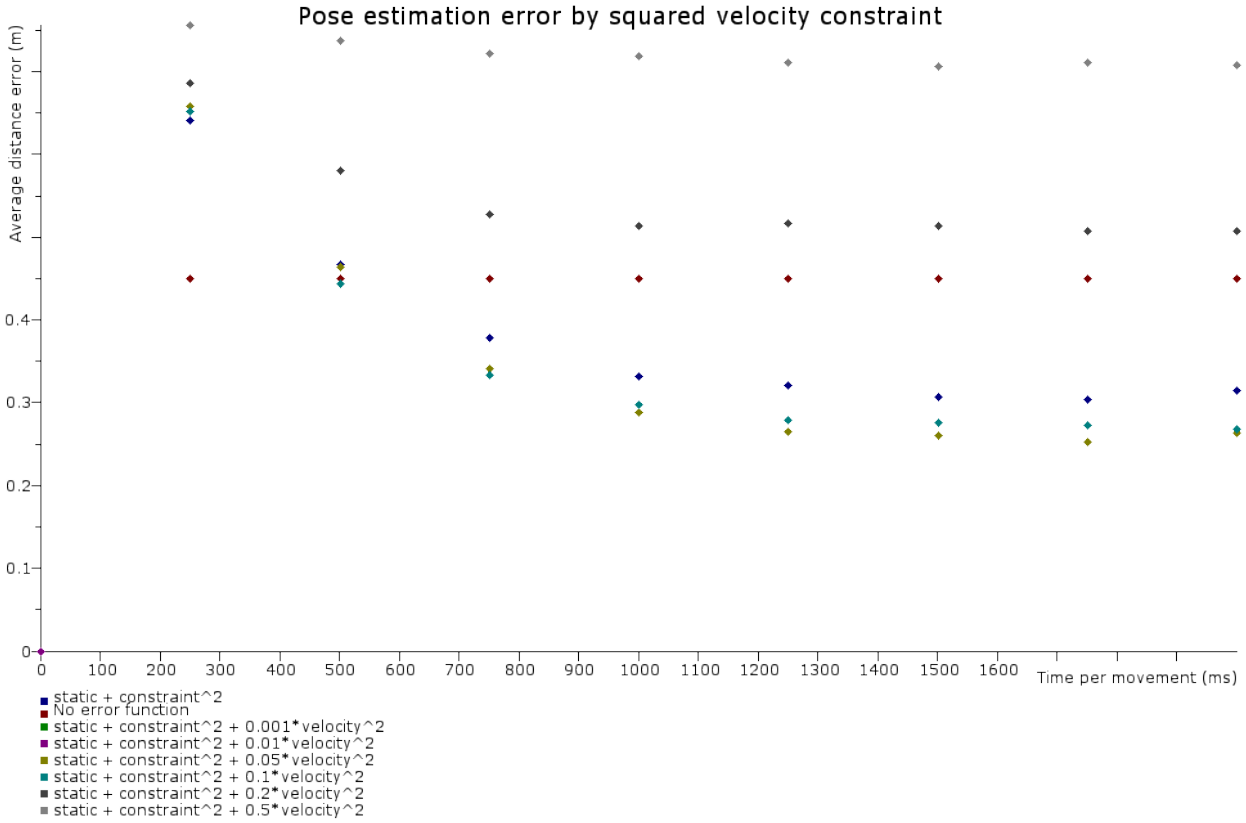

Runtime error estimation
Error (m)
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0
1
2
3
4
5
6
7
8
9
Velocity (m/s)
static^2
static^2 + constraint^2
static^2 + constraint
static + constraint^2
static + constraint

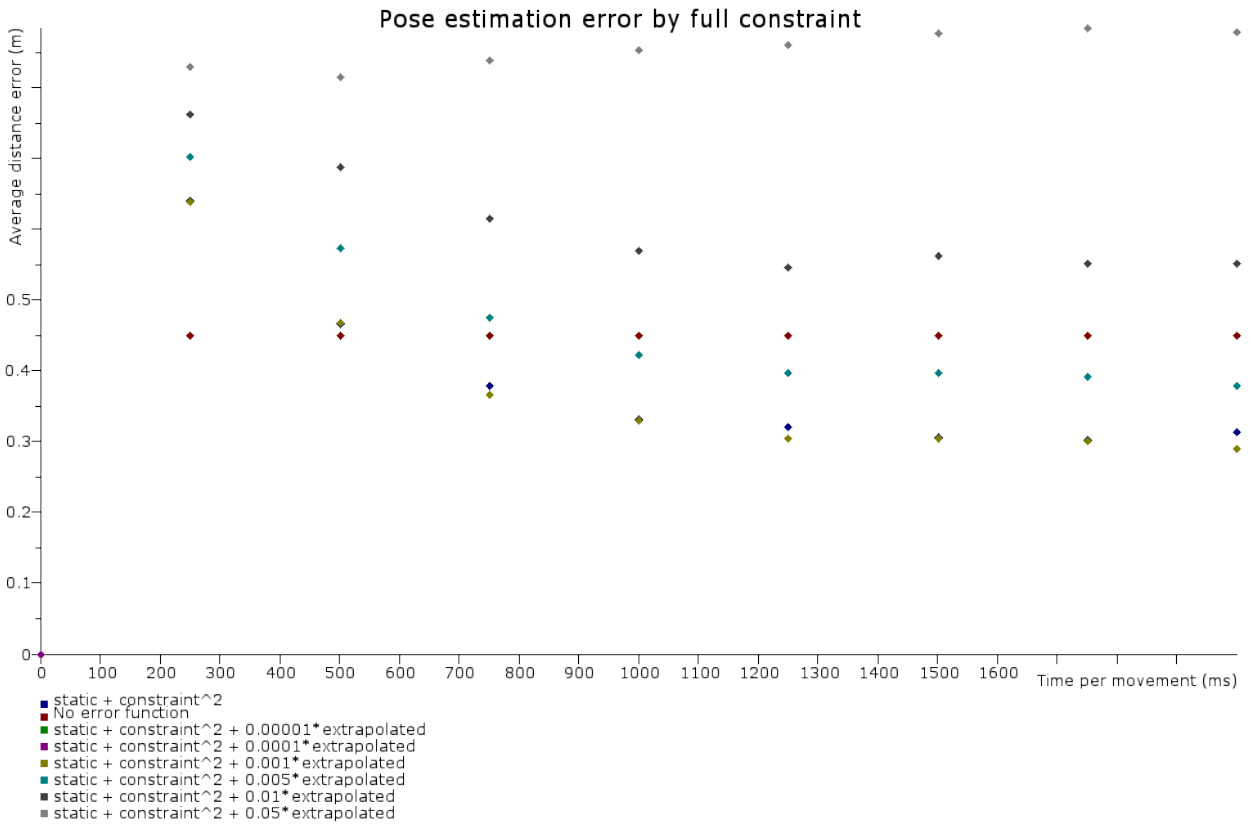
Performance with other error functions

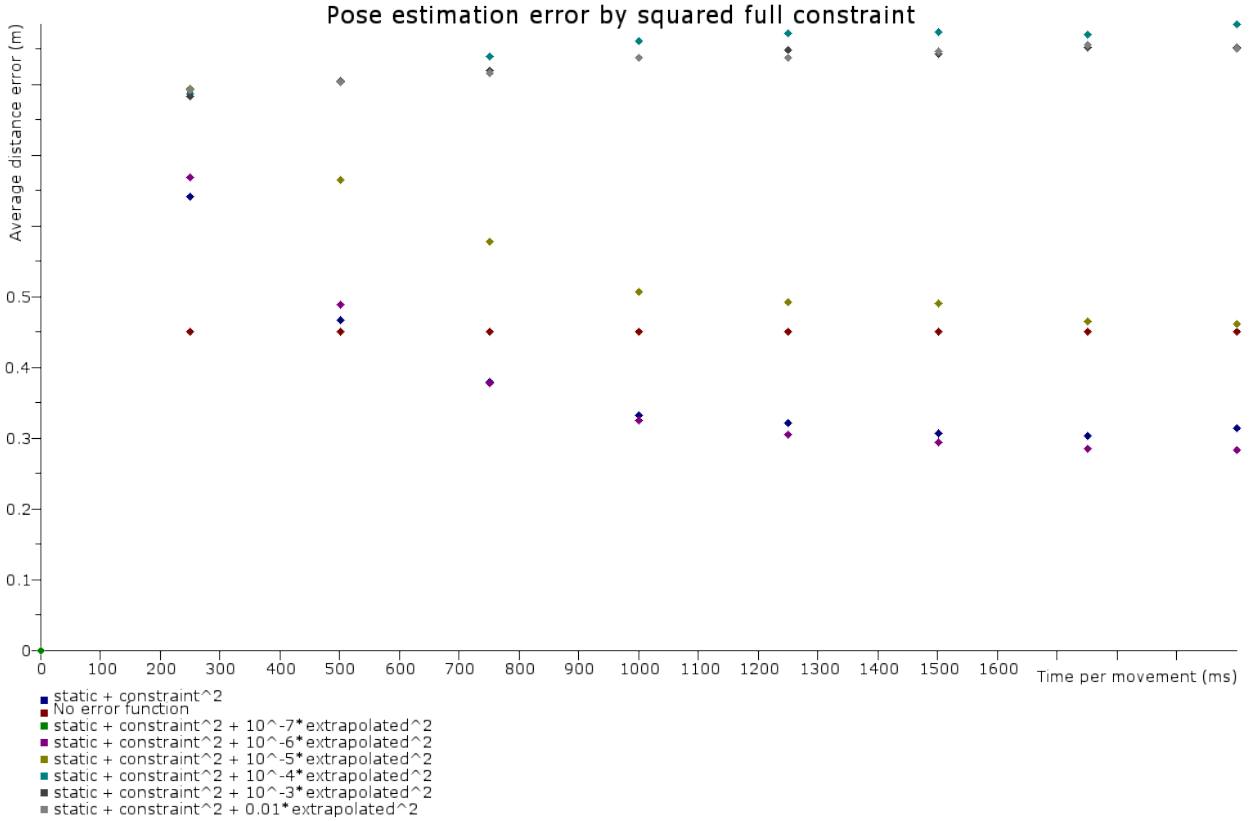




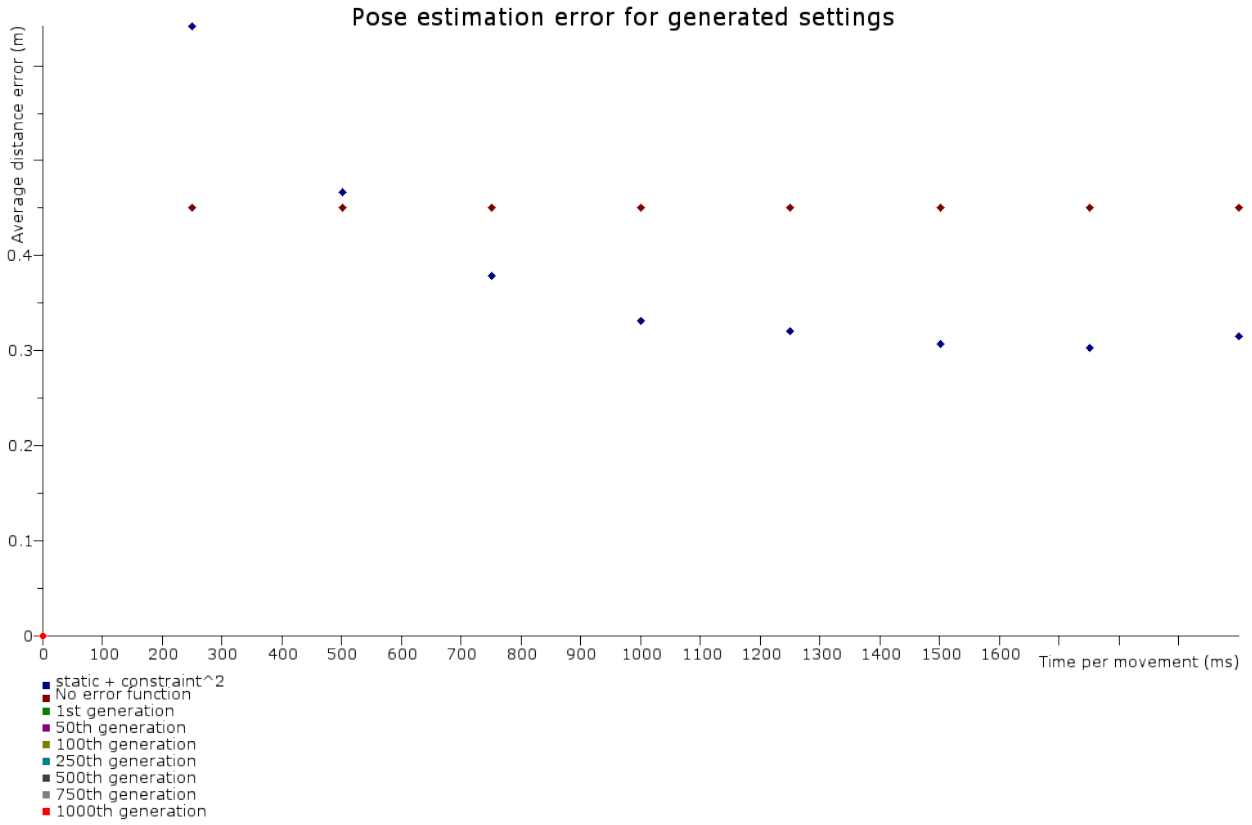

Pose estimation error by velocity constraint
Average distance error (m)
0
0.1
0.2
0.3
0.4
0 100 200 300 400 500 600 700 800 900 1000 1100 1200 1300 1400 1500 1600
Time per movement (ms)
static + constraint^2
No error function
static + constraint^2 + 0.05*velocity
static + constraint^2 + 0.1*velocity
static + constraint^2 + 0.2*velocity
static + constraint^2 + 0.5*velocity
static + constraint^2 + 1*velocity
static + constraint^2 + 5*velocity


Pose estimation error by squared velocity constraint
Average distance error (m)
0
0.1
0.2
0.3
0.4
0 100 200 300 400 500 600 700 800 900 1000 1100 1200 1300 1400 1500 1600
Time per movement (ms)
static + constraint^2
No error function
static + constraint^2 + 0.001*velocity^2
static + constraint^2 + 0.01*velocity^2
static + constraint^2 + 0.05*velocity^2
static + constraint^2 + 0.1*velocity^2
static + constraint^2 + 0.2*velocity^2
static + constraint^2 + 0.5*velocity^2


Pose estimation error by full constraint
Average distance error (m)
0
0.1
0.2
0.3
0.4
0.5
0 100 200 300 400 500 600 700 800 900 1000 1100 1200 1300 1400 1500 1600
Time per movement (ms)
static + constraint^2
No error function
static + constraint^2 + 0.00001*extrapolated
static + constraint^2 + 0.0001*extrapolated
static + constraint^2 + 0.001*extrapolated
static + constraint^2 + 0.005*extrapolated
static + constraint^2 + 0.01*extrapolated
static + constraint^2 + 0.05*extrapolated

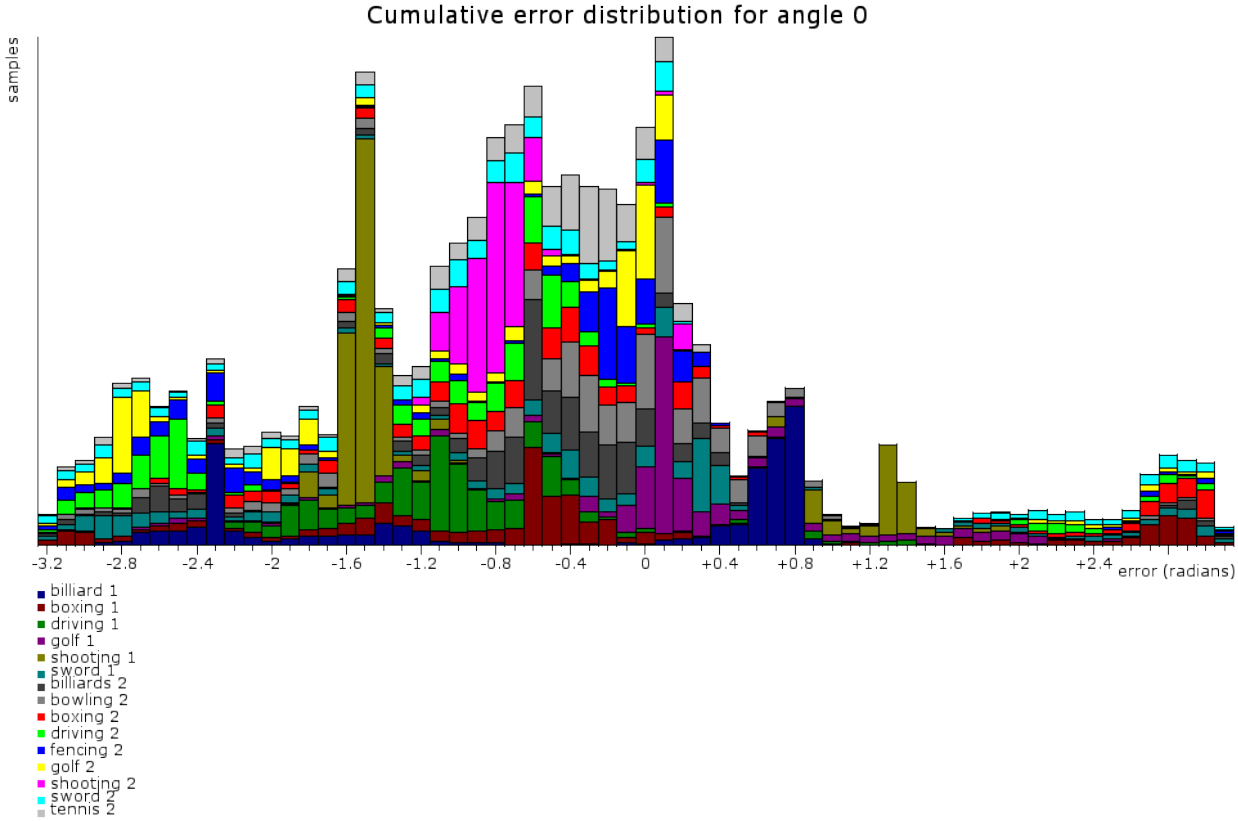

Pose estimation error by squared full constraint
Average distance error (m)
0
0.1
0.2
0.3
0.4
0.5
0 100 200 300 400 500 600 700 800 900 1000 1100 1200 1300 1400 1500 1600
Time per movement (ms)
static + constraint^2
No error function
static + constraint^2 + 10^-7*extrapolated^2
static + constraint^2 + 10^-6*extrapolated^2
static + constraint^2 + 10^-5*extrapolated^2
static + constraint^2 + 10^-4*extrapolated^2
static + constraint^2 + 10^-3*extrapolated^2
static + constraint^2 + 0.01*extrapolated^2

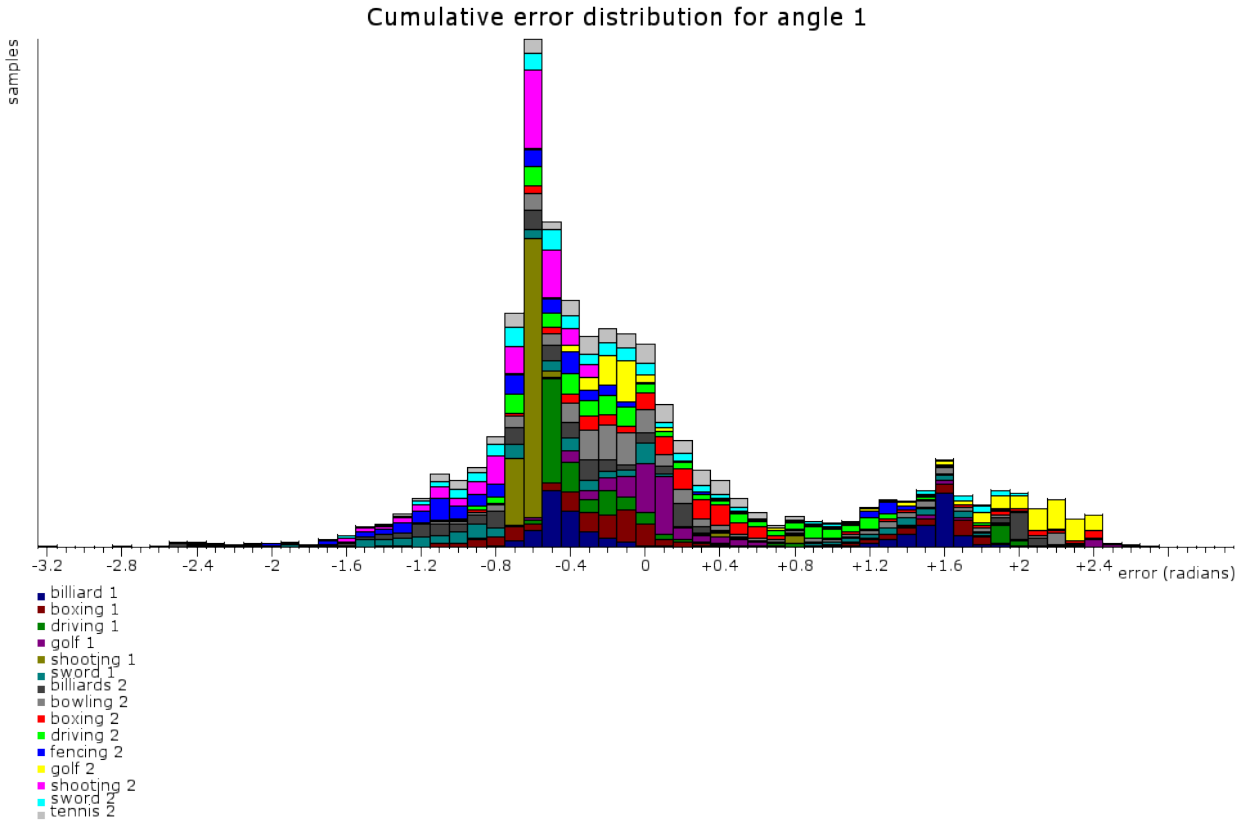
Performance after optimising settings

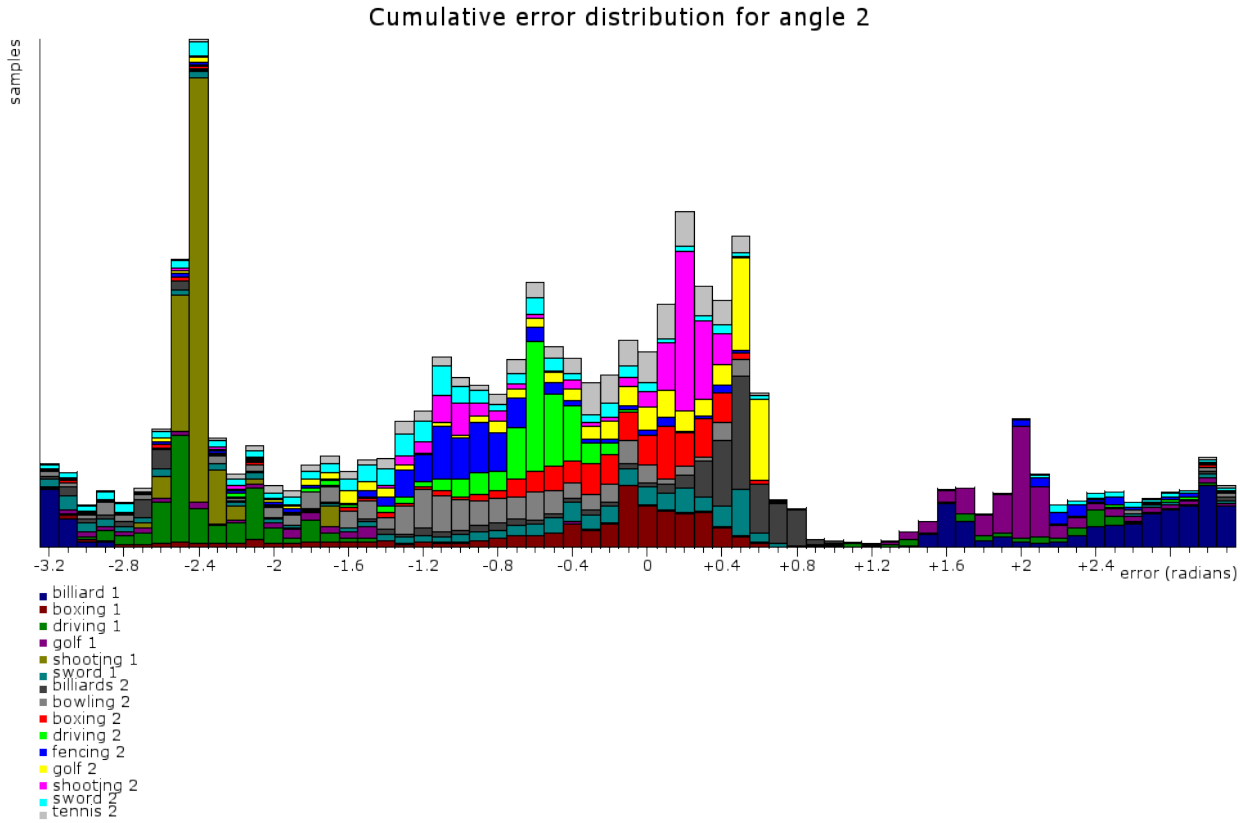


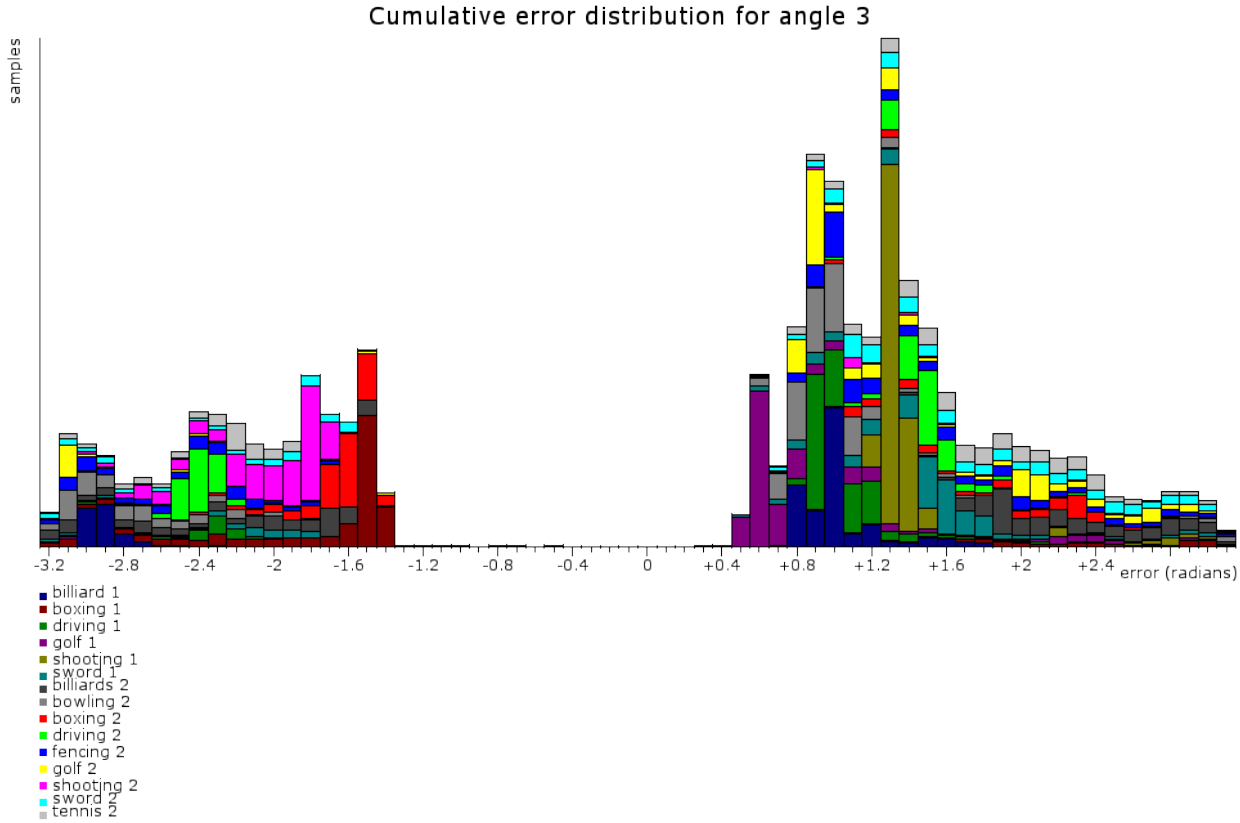
Appendix B – Raw results from the live sessions

Cumulative error distributions

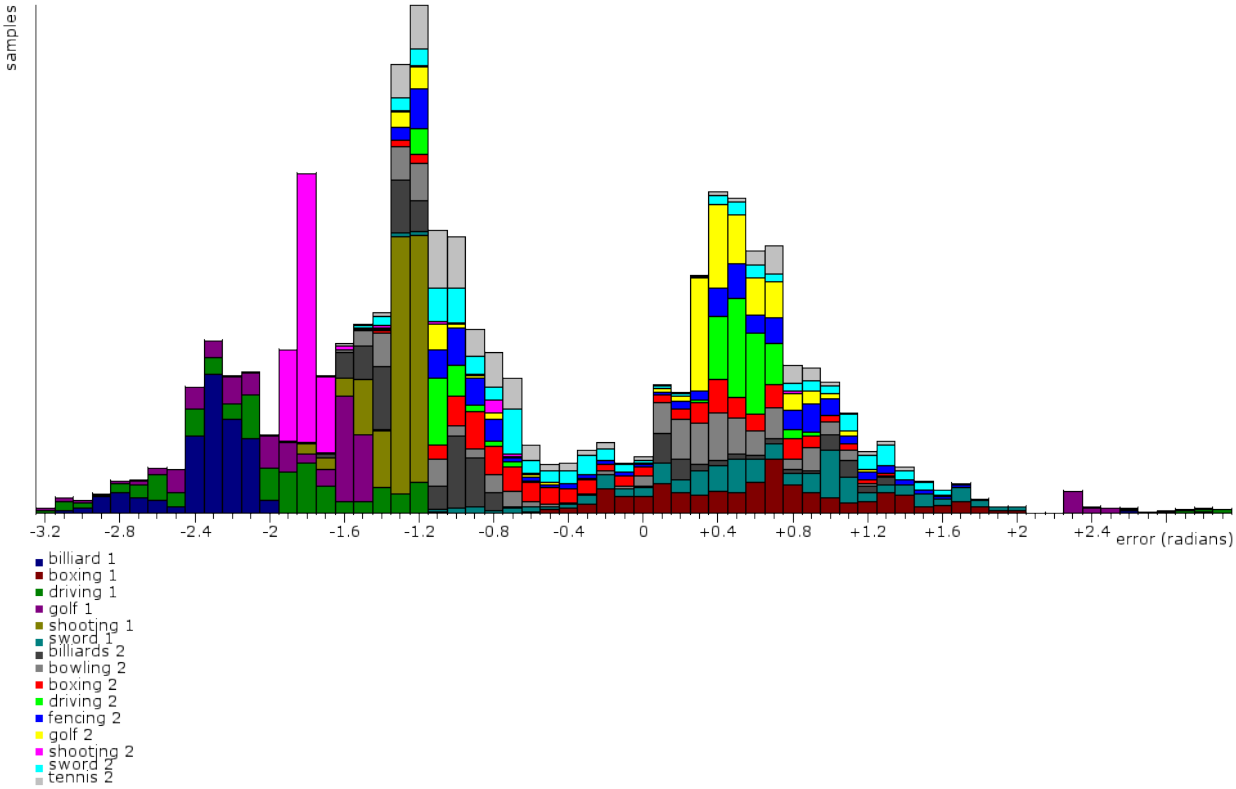




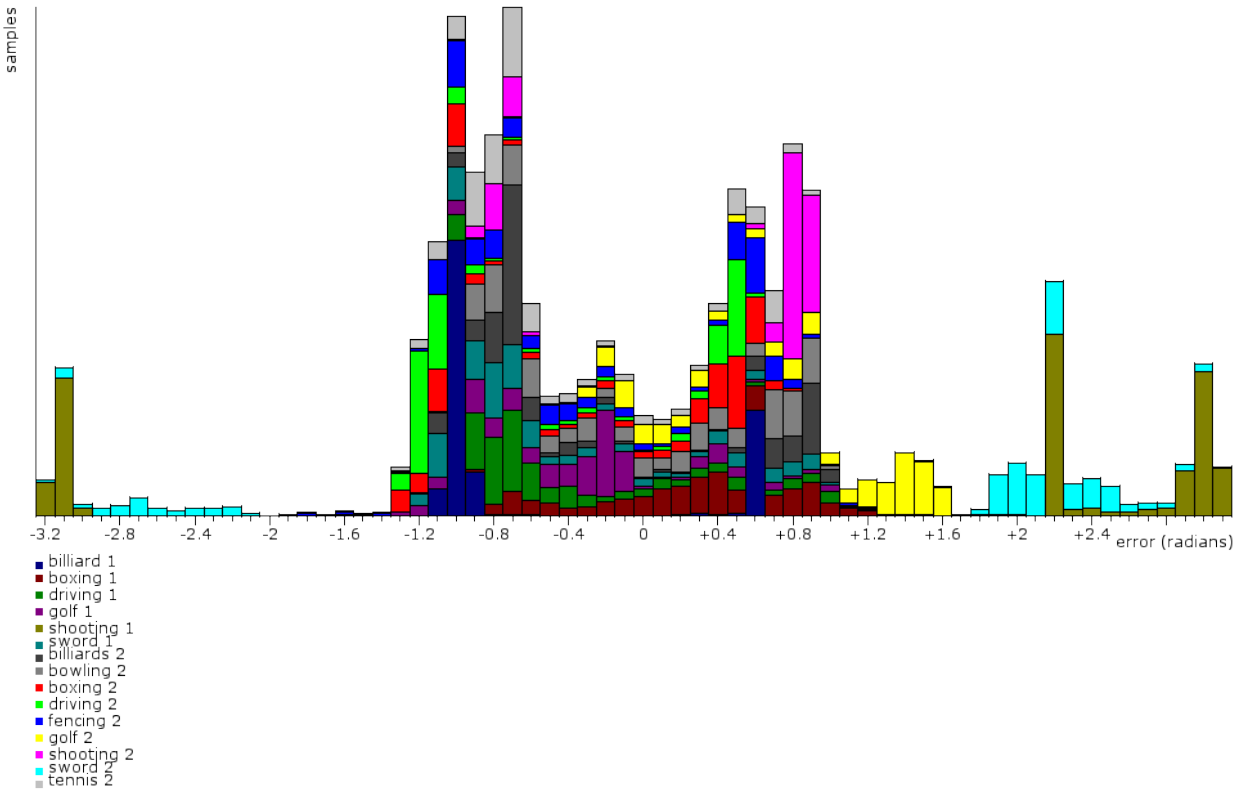

Cumulative error distribution for angle 2
samples
-3.2
-2.8
-2.4
-2
-1.6
-1.2
-0.8
-0.4
0
+0.4
+0.8
+1.2
+1.6
+2
+2.4
error (radians)
billiard 1
boxing 1
driving 1
golf 1
shooting 1
sword 1
billiards 2
bowling 2
boxing 2
driving 2
fencing 2
golf 2
shooting 2
sword 2
tennis 2


Cumulative error distribution for angle 3
samples
-3.2
-2.8
-2.4
-2
-1.6
-1.2
-0.8
-0.4
0
+0.4
+0.8
+1.2
+1.6
+2
+2.4
error (radians)
billiard 1
boxing 1
driving 1
golf 1
shooting 1
sword 1
billiards 2
bowling 2
boxing 2
driving 2
fencing 2
golf 2
shooting 2
sword 2
tennis 2

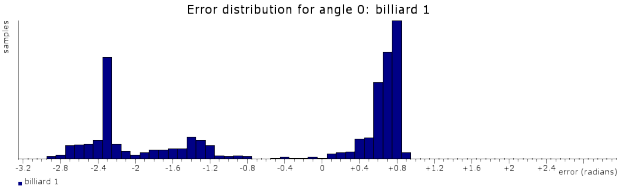
Cumulative error distribution for angle 4

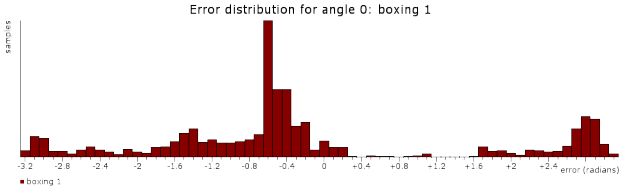


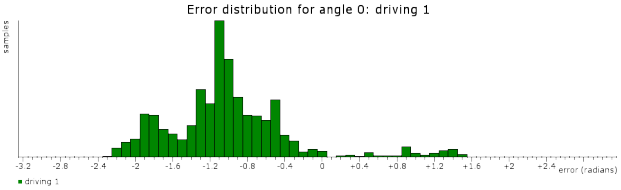
Cumulative error distribution for angle 5

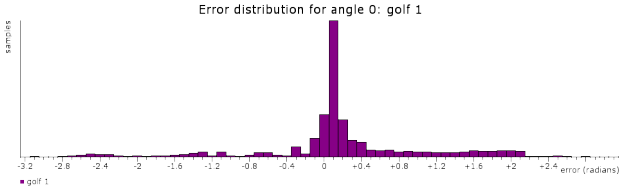


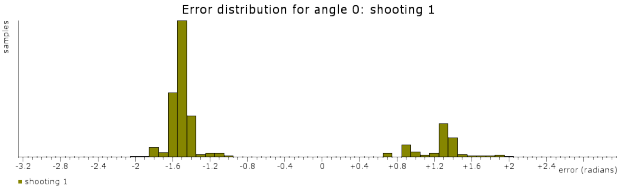
Error distributions for individual experiments

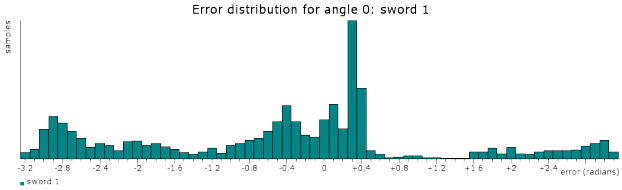


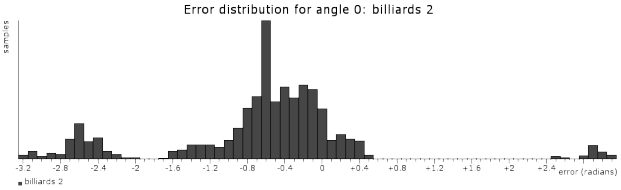


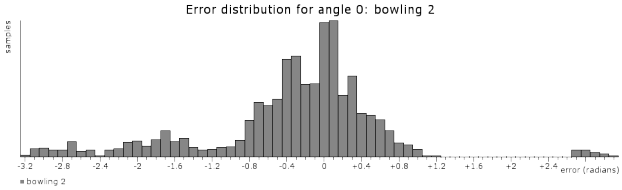


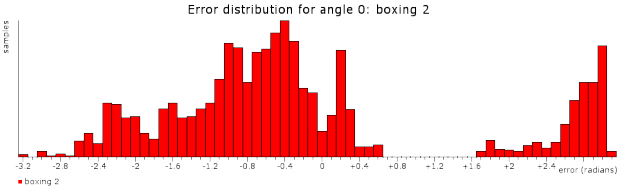


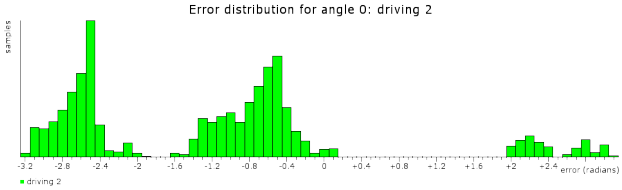


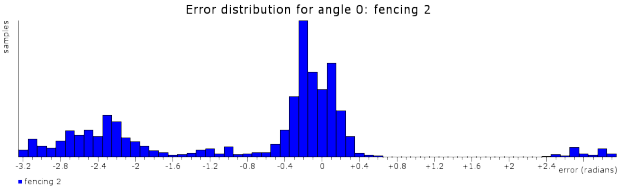


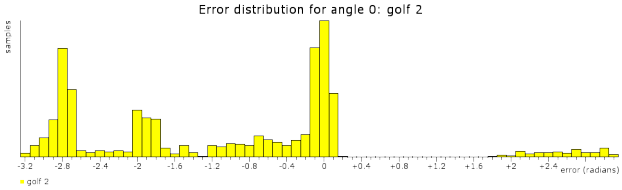


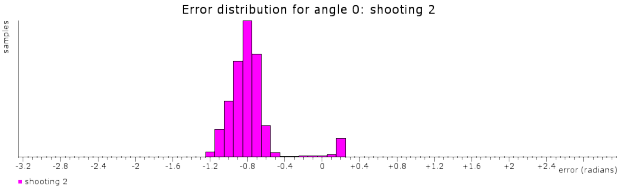


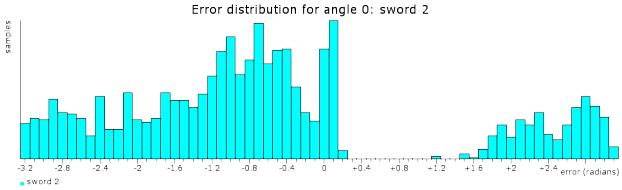


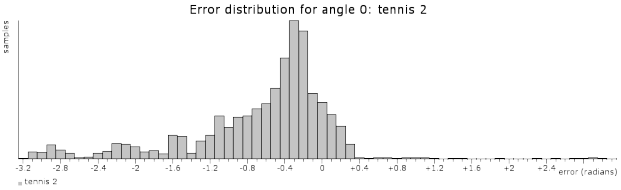


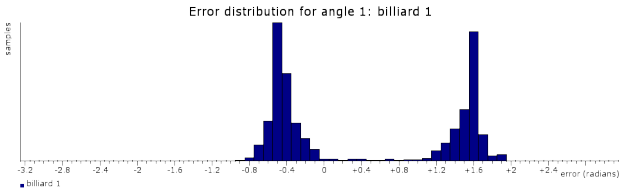


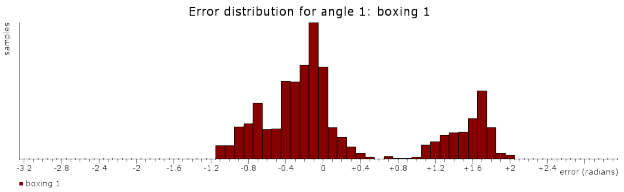


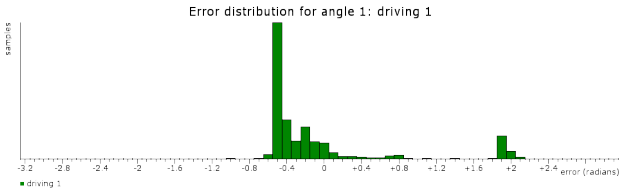


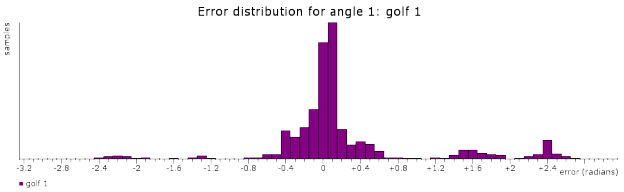


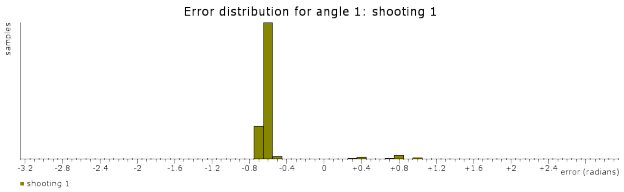


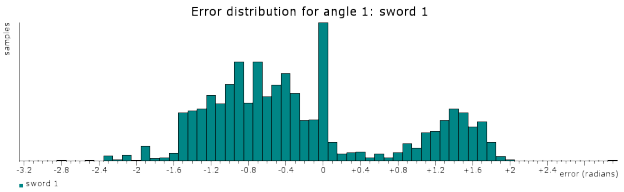

Error distribution for angle 1: billiard 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
billiard 1

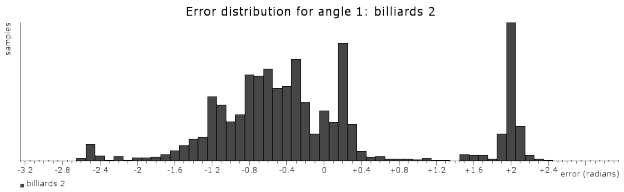

Error distribution for angle 1: boxing 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
boxing 1

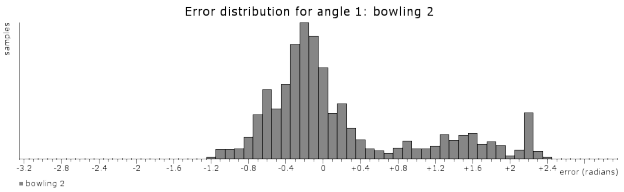

Error distribution for angle 1: driving 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
driving 1

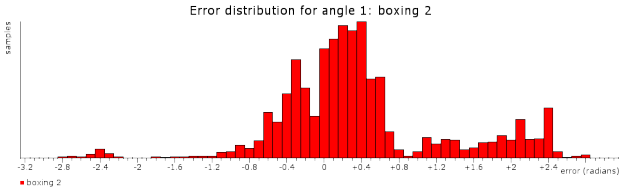

Error distribution for angle 1: golf 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
golf 1

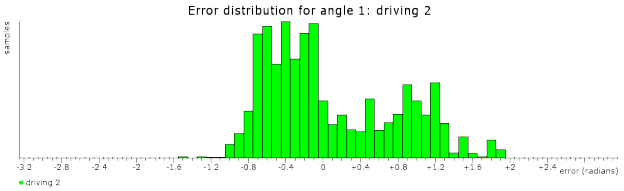

Error distribution for angle 1: shooting 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
shooting 1

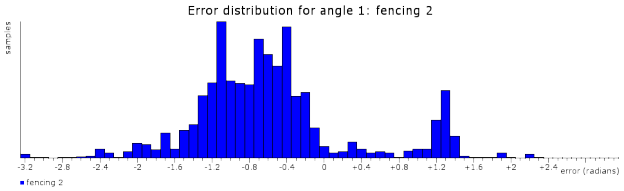

Error distribution for angle 1: sword 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
sword 1

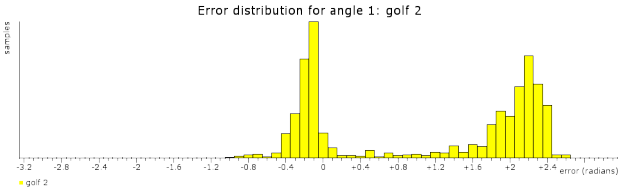

Error distribution for angle 1: billiards 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
billiards 2

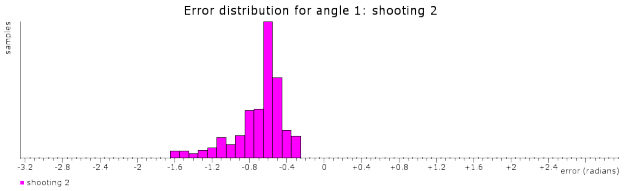

Error distribution for angle 1: bowling 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
bowling 2

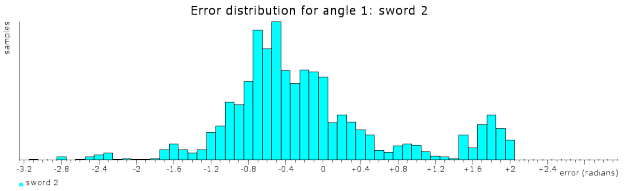

Error distribution for angle 1: boxing 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
boxing 2

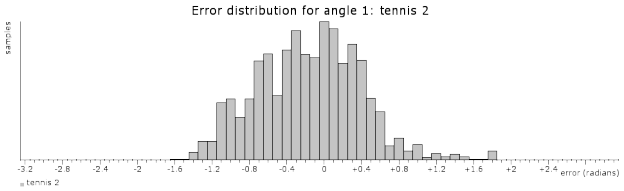

Error distribution for angle 1: driving 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
driving 2

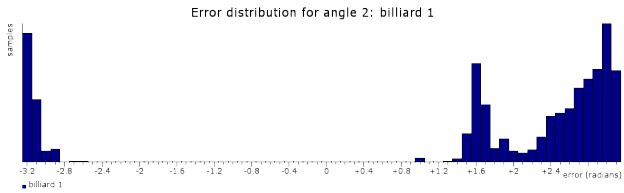

Error distribution for angle 1: fencing 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
fencing 2

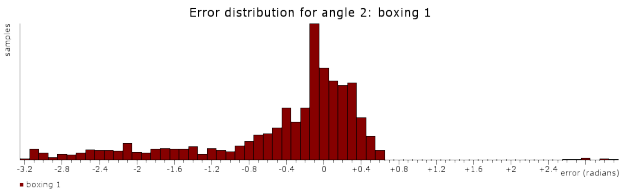

Error distribution for angle 1: golf 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
golf 2

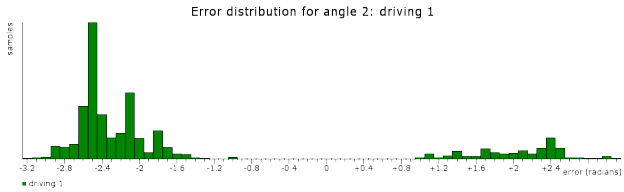

Error distribution for angle 1: shooting 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
shooting 2

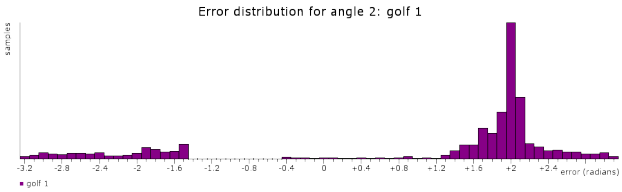

Error distribution for angle 1: sword 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
sword 2

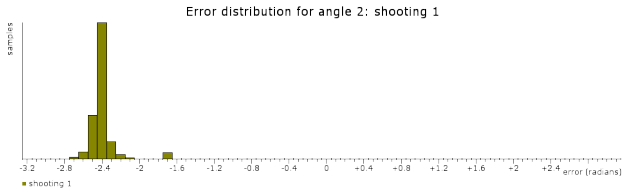

Error distribution for angle 1: tennis 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
tennis 2

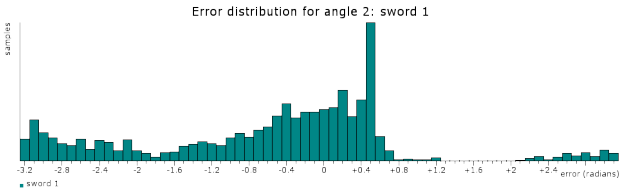

Error distribution for angle 2: billiard 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4 error (radians)
billiard 1

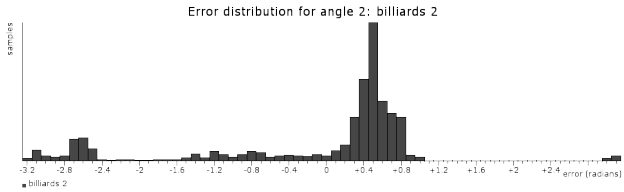

Error distribution for angle 2: boxing 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4 error (radians)
boxing 1

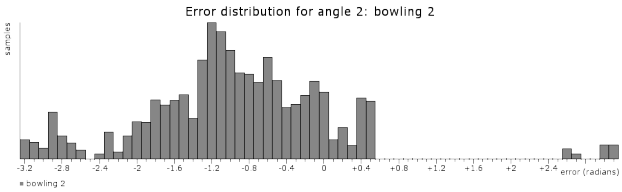

Error distribution for angle 2: driving 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4 error (radians)
driving 1

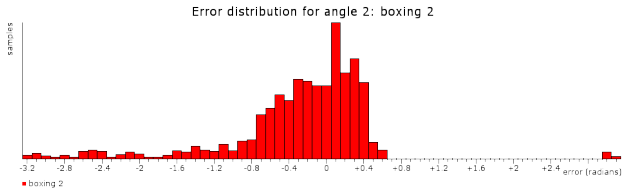

Error distribution for angle 2: golf 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4 error (radians)
golf 1

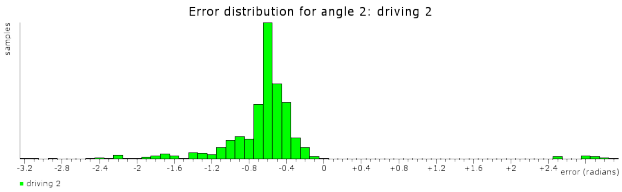

Error distribution for angle 2: shooting 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4 error (radians)
shooting 1

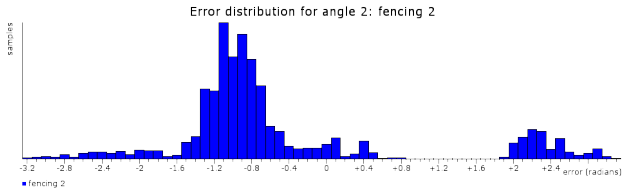

Error distribution for angle 2: sword 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4 error (radians)
sword 1

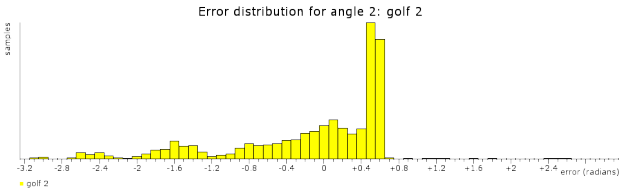

Error distribution for angle 2: billiards 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4 error (radians)
billiards 2

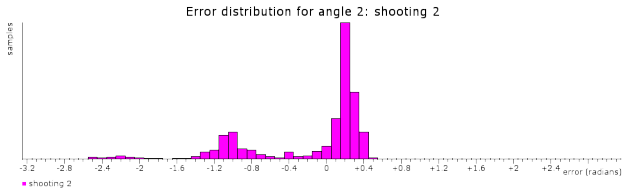

Error distribution for angle 2: bowling 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4 error (radians)
bowling 2

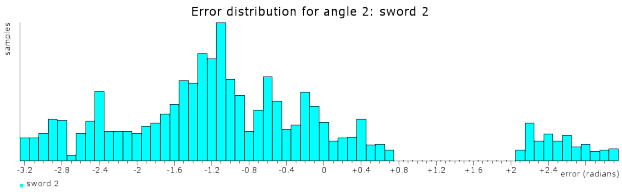

Error distribution for angle 2: boxing 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4 error (radians)
boxing 2

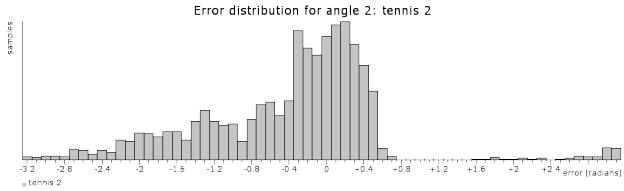

Error distribution for angle 2: driving 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4 error (radians)
driving 2

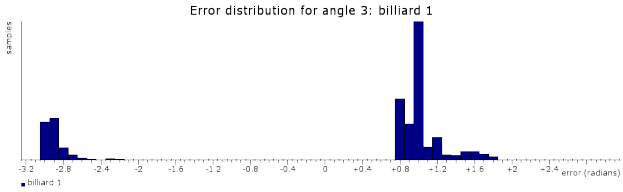

Error distribution for angle 2: fencing 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4 error (radians)
fencing 2

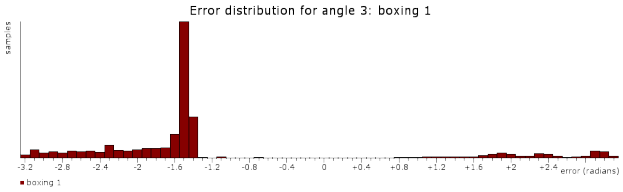

Error distribution for angle 2: golf 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4 error (radians)
golf 2

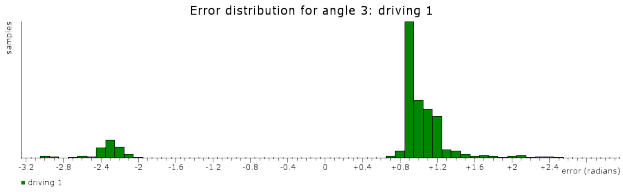

Error distribution for angle 2: shooting 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4 error (radians)
shooting 2

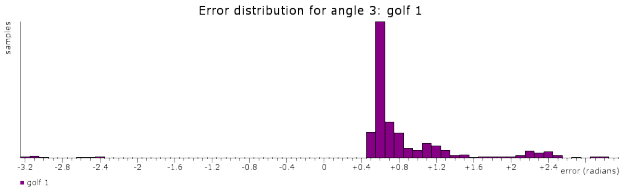

Error distribution for angle 2: sword 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4 error (radians)
sword 2

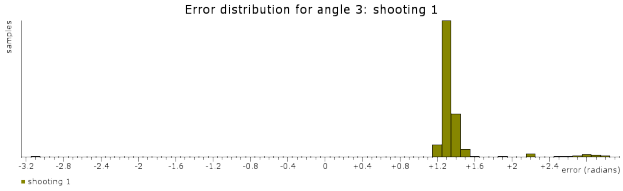

Error distribution for angle 2: tennis 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4 error (radians)
tennis 2

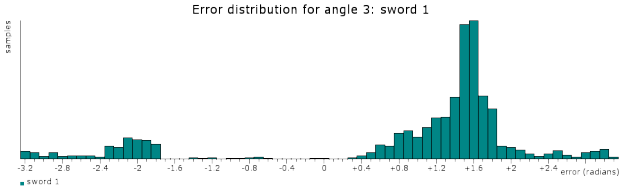

Error distribution for angle 3: billiard 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
billiard 1

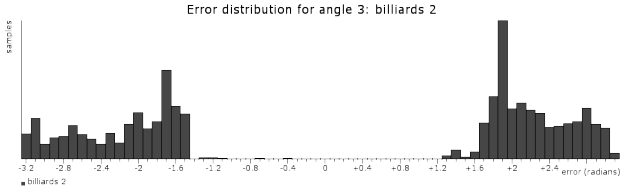

Error distribution for angle 3: boxing 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
boxing 1

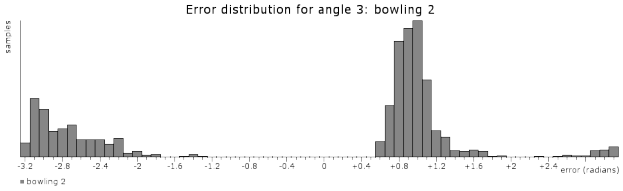

Error distribution for angle 3: driving 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
driving 1

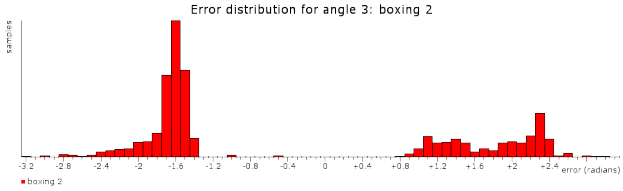

Error distribution for angle 3: golf 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
golf 1

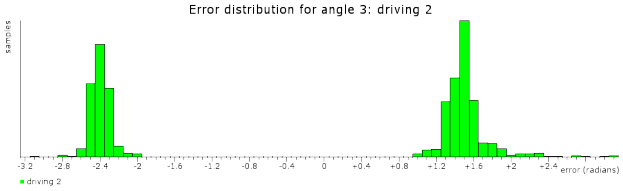

Error distribution for angle 3: shooting 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
shooting 1

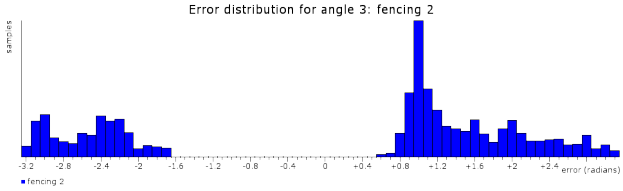

Error distribution for angle 3: sword 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
sword 1

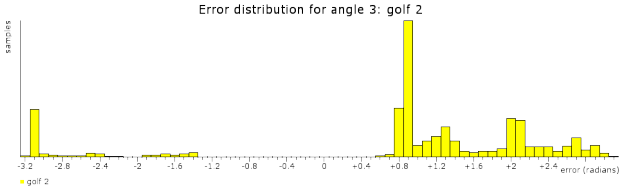

Error distribution for angle 3: billiards 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
billiards 2

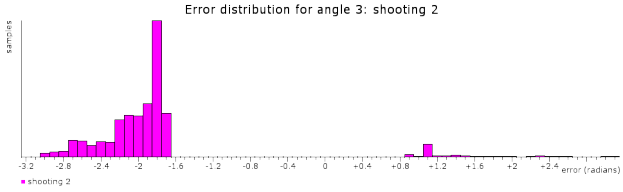

Error distribution for angle 3: bowling 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
bowling 2

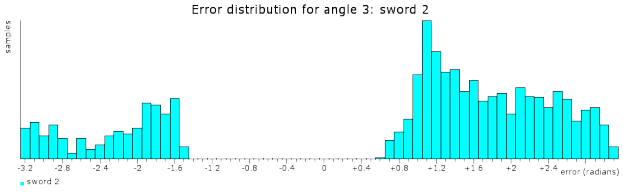

Error distribution for angle 3: boxing 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
boxing 2

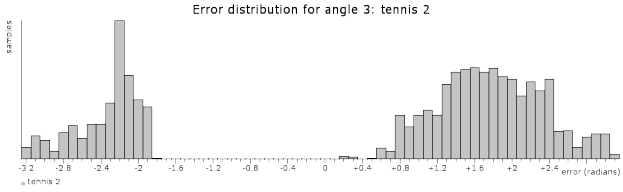

Error distribution for angle 3: driving 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
driving 2

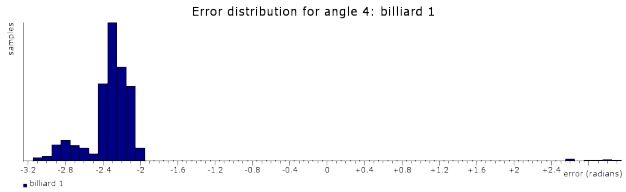

Error distribution for angle 3: fencing 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
fencing 2

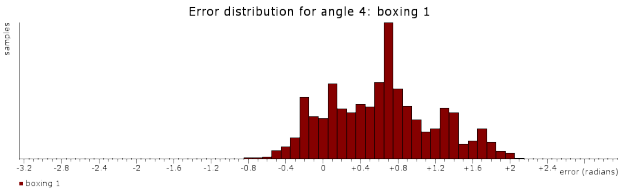

Error distribution for angle 3: golf 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
golf 2

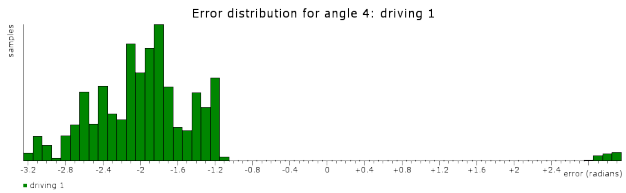

Error distribution for angle 3: shooting 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
shooting 2

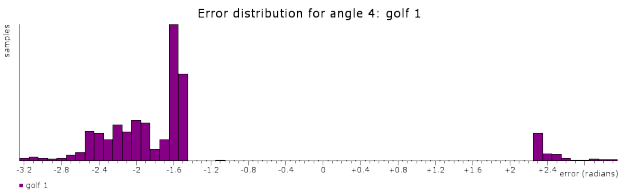

Error distribution for angle 3: sword 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
sword 2

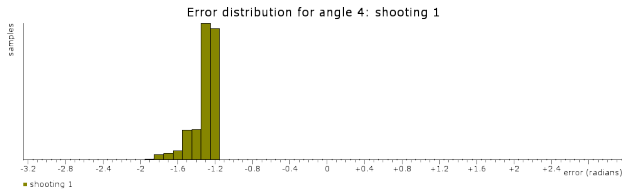

Error distribution for angle 3: tennis 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
tennis 2

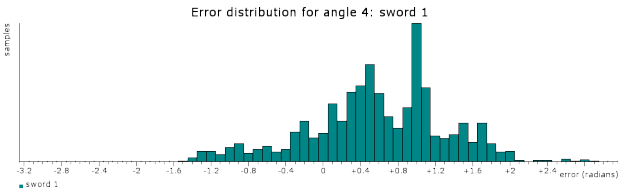

Error distribution for angle 4: billiard 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
billiard 1

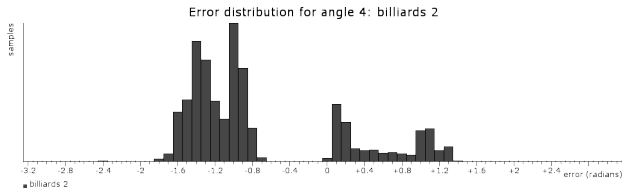

Error distribution for angle 4: boxing 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
boxing 1

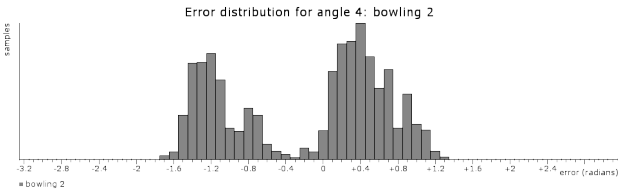

Error distribution for angle 4: driving 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
driving 1

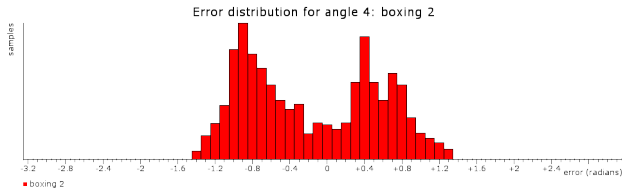

Error distribution for angle 4: golf 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
golf 1

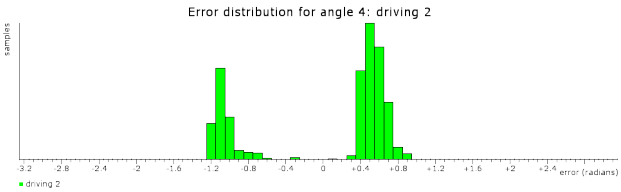

Error distribution for angle 4: shooting 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
shooting 1

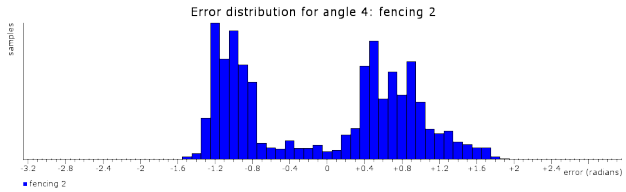

Error distribution for angle 4: sword 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
sword 1

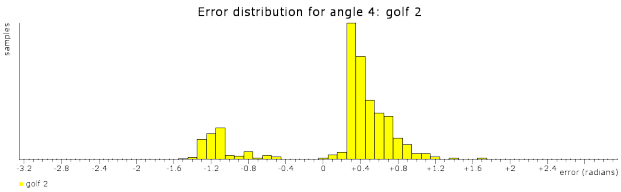

Error distribution for angle 4: billiards 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
billiards 2

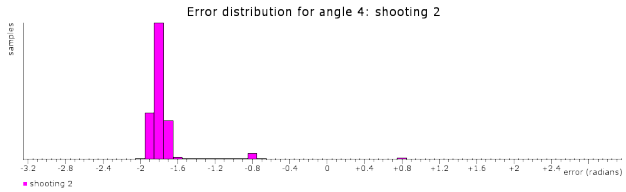

Error distribution for angle 4: bowling 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
bowling 2

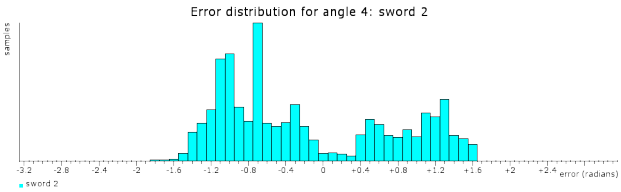

Error distribution for angle 4: boxing 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
boxing 2

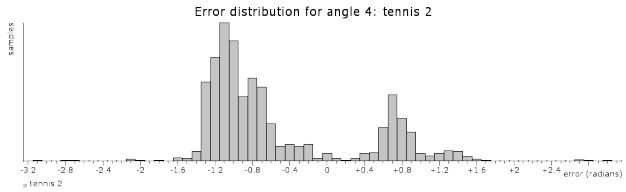

Error distribution for angle 4: driving 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
driving 2

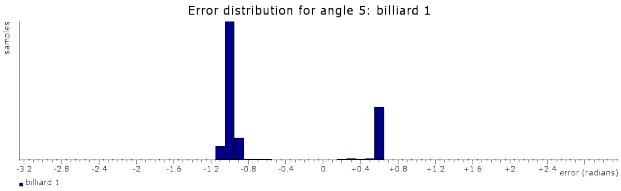

Error distribution for angle 4: fencing 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
fencing 2

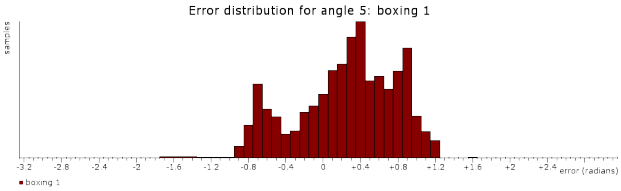

Error distribution for angle 4: golf 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
golf 2

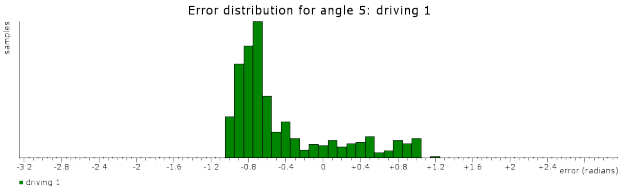

Error distribution for angle 4: shooting 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
shooting 2

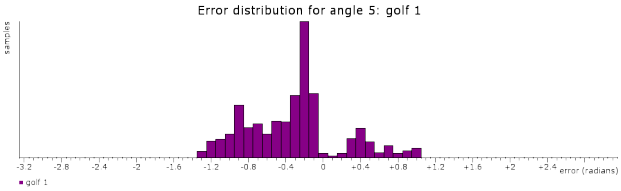

Error distribution for angle 4: sword 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
sword 2

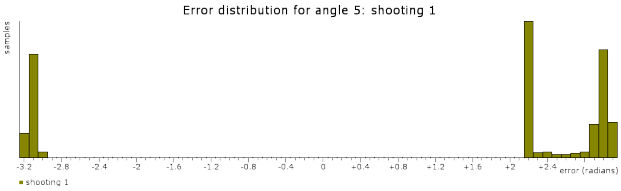

Error distribution for angle 4: tennis 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
tennis 2

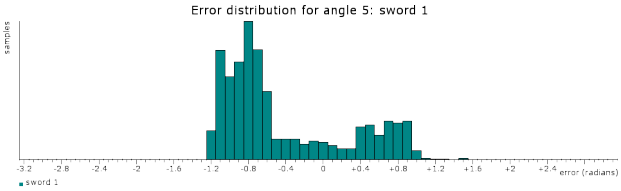

Error distribution for angle 5: billiard 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
billiard 1

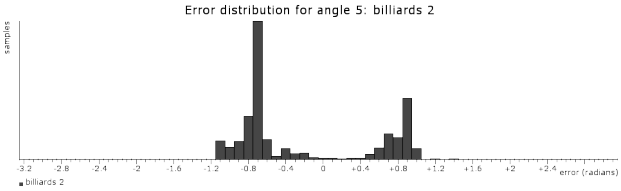

Error distribution for angle 5: boxing 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
boxing 1

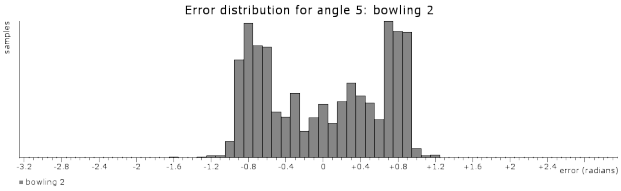

Error distribution for angle 5: driving 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
driving 1

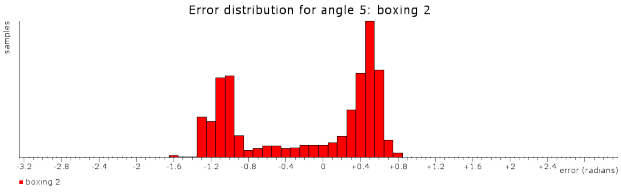

Error distribution for angle 5: golf 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
golf 1

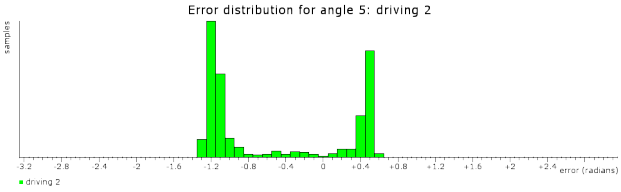

Error distribution for angle 5: shooting 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
shooting 1

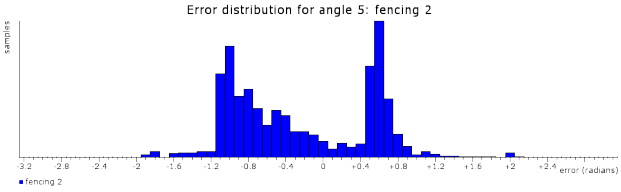

Error distribution for angle 5: sword 1
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
sword 1

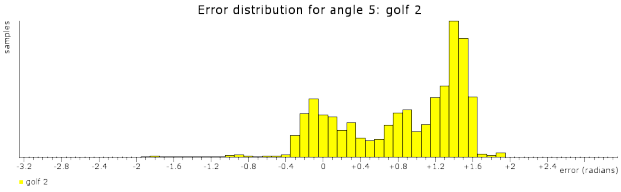

Error distribution for angle 5: billiards 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
billiards 2

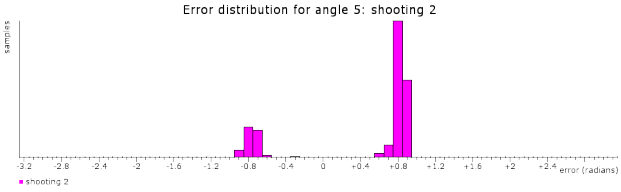

Error distribution for angle 5: bowling 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
bowling 2

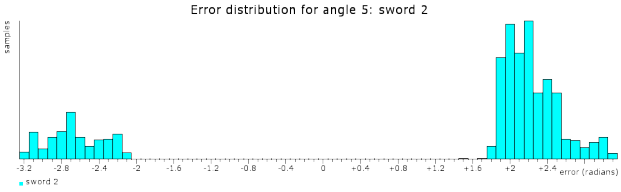

Error distribution for angle 5: boxing 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
boxing 2

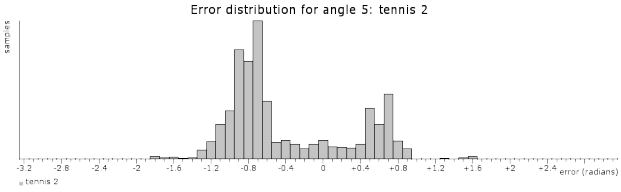

Error distribution for angle 5: driving 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
driving 2


Error distribution for angle 5: fencing 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
fencing 2


Error distribution for angle 5: golf 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
golf 2


Error distribution for angle 5: shooting 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
shooting 2


Error distribution for angle 5: sword 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
sword 2


Error distribution for angle 5: tennis 2
samples
-3.2 -2.8 -2.4 -2 -1.6 -1.2 -0.8 -0.4 0 +0.4 +0.8 +1.2 +1.6 +2 +2.4
error (radians)
tennis 2