# Method

Overview of the workflow. Each step gets described in detail during this chapter. The gray area is the process for each camera separately while in the blue area the different cameras are merged.

Overview of the workflow. Each step gets described in detail during this chapter. The gray area is the process for each camera separately while in the blue area the different cameras are merged.

## Calibration

TODO: de Absatz ned ahluege  
In order to work in three-dimensional space it is required to know how the pixels in a two-dimensional image relate to the three-dimensional world. For the camera model we consider the pinhole camera model for simplicity despite the fact that some cameras suffer from distortion. As mentioned in section [sec:tablecalibration] we were able to calibrate the cameras and got the corresponding camera matrix which transforms homogeneous 3D world coordinates to homogeneous 2D image coordinates.

### Camera Matrix

### Line Detection

## Player Detection

## 2D Pose Estimation

## 2D Refinement & mapping

## 3D Pose Fusion

The next goal is to determine the points in 3D space given its projections onto the multiple images from the different camera perspectives. By knowing the camera matrix of the camera projection from 3D to 2D we know that each point in the image corresponds to a line in 3D space and every point on this line get projected on the same point in the 2D image as mentioned in section [sec:calibration].

### Triangulation

The method to solve this problem is called triangulation. In theory with perfect measurements this method is trivial. So let’s assume we got a pair and of corresponding points in two different images. Remember that each point in the image belongs to a line in 3D space which are the green lines in the figure [fig:triangulation] . The lines intersect at which is the corresponding point in 3D space.

image [fig:triangulation]

But in practice the the image points are measured with arbitrary accuracy. This leads to the two points and in the figure [fig:triangulation]. If you consider the corresponding lines of these two points in 3D space (blue), they must not even intersect and if they do it is not correctly the corresponding 3D point . Reasons for the divergence may be noisy measurements or an inaccurate camera calibration which lead to wrong focal lengths and and therefore to a wrong projection .  
To find out which 3D point is the best estimate for the noisy measurements and you usually define an error measurement depending on and then minimize this error. Due to the fact that we do not always have all poses in 2D for the triangulation from the different views and the camera calibration is not error-free, I tried to use the Kalman Filter to do the triangulation. By constantly getting new measurements over time and the possibility of predicting the state further in time it seems to be a suitable application to be able to get the 3D poses.

### Extended Kalman Filter

The Kalman Filter (KF) is an iterative algorithm, based on the Bayesian Filter, that uses a series of measurements observed over time to find a good state estimation. The filter estimates the state using a form of feedback control where the measured values come into play. The theory in this section is based on the books *An Introduction to the Kalman Filter* and *Kalman and Bayesian Filters in Python* .

image [fig:kalmanprocess]

The algorithm works in a two-step process, as shown in figure [fig:kalmanprocess], which is repeated for every time step or more precise for every frame. In the prediction step, the Kalman Filter projects the current state variable forward in time using the process model and projects the error covariance ahead. In the second step, called the update step, the previously estimated state variable gets updated using the measurement residual weighted with the so called Kalman gain. Additionally the error covariance is updated. and a new external measurement as described in section x.

In order to update the state variable the residual is needed which is calculated from the current state variable and the newly added measurement . But the state variable is not part of the measurement space, therefore we have to project the state variable to the measurement space. This projection is basically a camera projection and is not linear due to a division, which is the reason why we use the Extended Kalman Filter (EKF). The reason why we work in measurement space and not in state space is because the measurements are not simply invertible which is exactly the goal of the EKF to fuse the 2D measurements into the 3D space.

### Designing the EKF

In the figure [fig:kalmanprocess] and in the following text the notation from the book *An Introduction to the Kalman Filter* slightly adapted to the filterpy notation will be used. We will omit the time subscript for the definitions of multiple variables in this section for the sake of clarity. For coordinates the subscript is used for the 3D space and for the 2D space of the -th camera. The state and measurement variable are written in bold to distinguish from the and coordinates. The last note is about the camera matrices which change with every frame. Thus, whenever the camera calibration is mentioned, one always refers to the current calibration.

#### State and Measurement

The EKF is designed to track one person using multiple cameras. The number of cameras is defined in the initialization. As mentioned in section [sec:poseestimation] the OpenPose model *COCO* consists of 15 keypoints. For the -th keypoint the corresponding part of the state vector includes the position in 3D space and their velocities:

All 18 keypoints together form the state vector:

The measurement vector includes for the -th keypoint the image coordinates for all cameras:

The complete measurement vector is again for all 18 keypoints:

#### Process Model

For the state propagation in the Predict step we use a simple linear model that does not change over time. For the sake of simplicity we assume constant velocity. For several reasons this is not the best choice for modeling the real world. The players move in different directions, the head keypoints behave completely differently to the hand keypoints etc. But due to the high frame rate one can neglect these aspects because the movement from one frame to the next is minimal. The linear motion model for the coordinates in 3D space is as follows . The time step represents the time between two frames. The resulting state transition model propagates the state vector for the -th keypoint further in time,

so the resulting complete state propagation matrix is a diagional matrix expressed using multiple submatrices :

This results in the first equation of the extended Kalman filter to project the state ahead:

Together with the next equation [eq:covariance], those two form the prediction step of the EKF. The estimation error covariance gets projected from the previous time step to the current time step .

At the begin the covariance matrix have to be initialized and afterwards the filter takes care of updating its value. We know that the position and velocity are correlated but due to the fact that we use constant velocity in our model we just initialize to zero. The matrix is a square matrix of the same dimension as the state vector.

#### Noise

In both equation [eq:covariance] and [eq:covariance2] we add a covariance matrix derived from white Gaussian noise. is a square matrix of the same dimension as the state vector and is called the process noise covariance. The matrix is also a square matrix with the dimension of the measurement vector and is called the measurement noise covariance. They’re called white Gaussian noise because they are independent of each other with normal probability distribution. During the EKF iteration they stay constant with values and :

The reason for the higher process noise is the inaccurate process model with constant velocity. So the filter tries to rely more on the measurements than on the prediction as desired.

#### Kalman gain and Error covariance

The residual gets added to the predicted state variable as described in the next section. But the residual is weighted by the Kalman gain. Higher values for the Kalman gain indicate that we give more trust to the measurement and a lower values indicate that we rely more on the prediction. So if the measurement uncertainty is smaller with respect to the error covariance we rely more on the measurement. The computation for the Kalman gain takes place in the Measurement Update step:

Another equation in the Measurement Update step is the update of the error covariance .

In contrast to and which are both constant, the error covariance and the Kalman gain will stabilize quickly and then remain constant as described in . These two equations involves the matrix which is explained in the section [sec:jacobian] below.

#### Measurement Model

The measurement model is designed through a function which is responsible to relate the state vector to the measurement . As we mentioned in section [sec:EKF] this function projects the state variable into the measurement space in order to be able to compute the residual which leads to this equation for the Measurement Update step:

The explanation how the function looks like and what the related matrix is will follow in detail. The filterpy implementation of the EKF requires for the update step the function with the arguments and the camera matrices for all cameras. The output of this function is a vector which looks exactly like the measurement vector but do not get confused, it is just the same space and no new measurement vector:

As mentioned in section [sec:cameramatrix] a homogeneous point in the 3D space can be projected to the image screen using the camera matrix . I will explain this step for the coordinates of the state vector for one keypoint to map to the corresponding measurement space of one specific camera . I will omit the time subscript in this section for the sake of understanding.

In order to get the correct scale and dimension for the screen coordinates one have to divide the result from equation [eq:homogenous] by and drop the last entry for the coordinate. This forms a part of the function :

To make it clear once again, this was just to transform the 3D coordinates for one keypoint into the image space of one camera. So for the -th keypoint of the state vector we drop the velocities and transform the 3D coordinates to screen coordinates for each of the cameras. This gives us a bigger part of :

Finally we can formulate the whole function for all 18 keypoints:

#### Jacobian Matrix H

By the fact that the relationship between state space and the measurement space is a nonlinear function we need to linearize it by evaluating its partial derivatives with respect to the state vector like a Taylor series. The resulting matrix is the Jacobian matrix . How the derivations of one screen coordinate pair (the -th keypoint) like in equation [eq:screencoordinates] for the camera looks like is explained as follows:

The matrix would be much wider because of the partial derivatives for the other state variables but they are all zero so we just note this block matrix and build the Jacobian with multiple similar blocks. So the whole corresponding block of the Jacobian for the -th keypoint with respect to all cameras is presented:

And finally we can formulate the entire Jacobian with the partial derivatives for the entire state vector, note that in the steps above the subscript has been removed for readability.

Like the function the filterpy implementation also requires the Jacobian of as input for the update step. So the Jacobian is also a function with the same arguments as the function , namely the state vector and the camera matrices. For the readability we first define some variables which are used afterwards:

So what’s left are the partial derivatives. The computation for those from equation [eq:partial] are shown with respect to the -th keypoint and the -th camera, the others are analog but with the corresponding keypoint coordinates and the corresponding camera matrices.

#### State Initialization

TODO

All together result in the final EKF. For a good overview the five equations for the filter are repeated here:

**The Update (“Predict”):**

(1) Projet the state ahead:

(2) Project the error covariance ahead:

**Measurement Update (“Correct”):**

(1) Compute the Kalman gain:

(2) Update estimate with measurement :

(3) Update the error covariance: