Tampereen yliopisto

Miro-Markus Tuokko

3D TRACKING OF A MOBILE DEVICE

Master of Science Thesis
Faculty of Engineering and Natural Sciences
Risto Ritala
Heikki Huttunen
August 2020

ABSTRACT

Miro-Markus Tuokko: 3D tracking of a mobile device
Master of Science Thesis
Tampere University
Automation Engineering
August 2020

There are multiple different methods for tracking the location and orientation of a rigid object. Finnish company Piceasoft Ltd. has a need to track a mobile phone while it is being turned by hand. The tracker needs to be running on an Android phone. In this thesis, different tracking methods are studied. How well the methods would fit this application is evaluated. The edge-based tracking method is chosen as the most fitting because it does not require a training stage, and the same tracker can be used for tracking different mobile phones. The edge-based method is also light enough so it can be executed on a mobile device.

Mathematical concepts that are needed in the implementation of an edge-based tracker are discussed in detail. The first step in the tracking process is to project a 3D model of the object to the image plane. Internal and external matrices are used in perspective projection. The outermost edges of the projected model are found next because the inner edges are not used by the tracker. The outermost edges are divided into control points.

The image from the camera is used to find the corresponding points for the control points. A search line is drawn for each one of the control points. The search lines are perpendicular to the outermost edges of the model. A measurement point should be located on each of the search lines. The measurement points are located on the edges of the phone that is in the image. The search lines are rotated and stacked on top of each other to create a search bundle. Histograms are used to find the measurement points from the search bundle.

A new estimate for the pose can be calculated from the control points and the corresponding measurement points. The pose estimate is calculated iteratively using the Gauss-Newton algorithm. The 3D model can then be projected to the image plane using the new pose estimate, and the process can be repeated.

The performance of the implemented tracker is evaluated. The tracker can perform the original task that it was supposed to do, but not in all conditions. The tracking may fail if the background is the same color as the phone that is being tracked. Edges in the image that are parallel and close to the edges of the phone may also cause tracking failure. How the phone is grabbed while it is being turned also matters. The tracking may fail if a large part of the phone is not visible to the camera. The phone must not move too much between consecutive processed frames, or the measurement points cannot be found from the next frame, and the tracking fails.

Keywords: 3D, tracking, edge, pose, mobile, device

TIIVISTELMÄ

Miro-Markus Tuokko: Mobiililaitteen asennon määrittäminen kamerakuvasta
Diplomityö
Tampereen yliopisto
Automaatiotekniikka
Elokuu 2020

Jäykän esineen asennon määrittämiseen kuvan perusteella on kehitetty useita menetelmiä. Suomalaisella Piceasoftilla on tarve pystyä määrittämään mobiililaitteen asento kamerakuvasta, samaan aikaan kun laitteesta otetaan kädellä kiinni ja se käännetään ympäri. Asennon tunnistava ohjelmisto suoritetaan Android-mobiililaitteella. Tässä työssä käydään lävitse erilaisia asennon määrittämiseen soveltuvia metodeja, ja niiden soveltuvuutta tässä työssä ratkaistavaan ongelmaan arvioidaan. Reunoihin perustuva metodi valitaan parhaaksi metodiksi tähän työhön, koska se ei tarvitse erillistä koulutusvaihetta. Reunoihin perustuvaa metodia voidaan myös käyttää erilaisten puhelinten asennon seuraamiseen. Metodi on myös tarpeeksi kevyt suoritettavaksi mobiililaitteella.

Matemaattiset konseptit, joita tarvitaan reunoihin perustuvan metodin toteuttamiseen, käydään työssä läpi. Asennon määrittämisprosessin ensimmäinen vaihe on projisoida seurattavan esineen 3D-malli kuvatasolle. Perspektiiviprojektiossa käytetään kameran sisäistä ja ulkoista matriisia. Seuraava vaihe on etsiä projisoidun mallin uloimmat reunat, koska metodi ei hyödynnä sisempiä reunoja. Uloimmat reunat jaetaan kontrollipisteisiin.

Kameralta saatavaa kuvaa käytetään kontrollipisteitä vastaavien mittauspisteiden löytämiseen. Jokaiselle kontrollipisteelle piirretään etsintäviiva. Etsintäviivat ovat kohtisuorassa uloimpiin reunoihin nähden. Jokaisella etsintäviivalla tulisi olla yksi mittauspiste. Mittauspisteet sijaitsevat kuvassa olevan puhelimen reunoilla. Etsintänipun muodostamista varten etsintäviivat käännetään niin, että ne ovat vaakatasossa. Etsintänippu muodostetaan pinoamalla käännetyt etsintäviivat päällekkäin.

Asennolle lasketaan uusi arvio käyttäen kontrollipisteitä sekä niitä vastaavia mittauspisteitä. Asentoarvio lasketaan iteratiivisesti käyttäen Gauss-Newton menetelmää. 3D-malli voidaan seuraavaksi projisoida kuvatasolle käyttäen uutta asentoarviota, ja prosessi voidaan toistaa uudelleen.

Työssä myös arvioidaan toteutetun asentoarvioijan suorituskykyä. Toteutettu ohjelma pystyy suoriutumaan alkuperäisestä tehtävästään, mutta ei kaikissa olosuhteissa. Asennon arviointi voi epäonnistua, jos tausta ja seurattava laite ovat samanväriset. Kuvassa olevat reunat, jotka ovat lähellä ja samansuuntaisia seurattavan laitteen reunojen kanssa, voivat aiheuttaa asennon arvioinnin epäonnistumisen. Myös sillä on merkitystä, miten seurattavasta laitteesta on otettu kädellä kiinni. Asennon arviointi saattaa epäonnistua, jos käsi peittää suuren osan laitteesta, eikä laite ole kokonaan näkyvissä kameralle. Laite ei saa myöskään liikkua liian paljon peräkkäisten käsiteltyjen kuvien välillä, koska mittauspisteiden etsintä epäonnistuu tällöin, mikä johtaa asennon arvioinnin epäonnistumiseen.

Avainsanat: 3D, asento, arviointi, mobiililaite

PREFACE

I want to thank Piceasoft for giving me the opportunity to make this Master of Science Thesis. The help, guidance, and feedback given to me by Henri Salminen has been a great help during the writing process. I want to thank Risto Ritala for making sure that my thesis is according to the scientific standards and Heikki Huttunen for being the second examiner of my work.

The support and encouragement from my family have been priceless to me through my studies. I want to thank my friends for the countless fun nights and for struggling with me through the courses. Special thanks to Elisa for being there for me during the writing process.

Tampere, 28th August 2020

Miro Tuokko

CONTENTS

LIST OF FIGURES

LIST OF SYMBOLS AND ABBREVIATIONS

AR	Augmented reality
N3M	Natural 3D Markers
SIFT	Scale-invariant feature extraction
DLT	Direct Linear Transform
PnP	Perspective-n-Point
POSIT	pose from orthography and scaling with iterations
LM	Levenberg-Marquardt
LAD	Least absolute deviations
RAPID	Real-time Attitude and Position Determination
RGB	Red, green, and blue
HSV	Hue, saturation, and value
FPS	Frames per second

1. INTRODUCTION

Mobile devices have become very popular, which has created a market also for second-hand devices. The price of new flagship models is increasing with every device generation. New generations offer more processing power and improved hardware, but new groundbreaking features have become less frequent. Consumers now have the option of choosing a second-hand mobile device instead of a new one. The main benefit of choosing a used device is the lower price compared to a new one.

The condition of the device has a major impact on the price of a second-hand device. To find a price acceptable both for the seller and the buyer, a reliable, easy, and fair way to evaluate the condition of a used mobile device is needed. Piceasoft Ltd, a Finnish company, provides a service for assessing the condition of mobile devices. The assessment includes testing the hardware and software of the device. As part of this assessment process, the amount of scratches on the screen and the body of the device is detected.

Scratches on the device can be detected only visually. A high-quality image of the device is needed for scratch detection. In order to find all the scratches on a mobile device, two images are required, one from the front side and another from the back. The device must be identified based on the camera video stream before the used device is imaged, to assure that the correct device is being imaged and assessed. Identifying the device from the front side is quite straightforward because the content on the screen of the device is known, e.g., a known QR code can be displayed on the screen and then read from the camera video stream.

Identifying the device while the backside is visible to the camera is more complicated. In this thesis, the identification is made by tracking the device while it is being turned by hand. In the beginning, the device is front side up, and a QR code is displayed on the screen. The QR code is read, and the system identifies the device. After that, the tracking of the device is started. The device that is being tracked is turned around by hand, so the backside becomes visible to the camera. The identity of the backside seen by the camera is known if the tracking is successful. Finally, a picture of the backside of an identified device can be taken for scratch detection.

This thesis focuses on the tracking phase of the process, so scratch detection is not discussed. The first research question of this thesis is:

What kind of tracking methods exists, and which one of them is the most suitable for tracking a mobile device while it is being turned around by hand?

Available tracking methods were studied to answer this question. The best method for this application needs to be chosen based on the pros and cons of the studied methods. The second research question of this thesis is:

How can the tracking method be implemented on an Android device, and in what kind of situations it will fail?

The tracker needs to be running on an Android device. The performance of the tracker is evaluated because it is essential to understand the limitations of the method.

The thesis is organized as follows. Section 2 reviews the relevant pose tracking methods. The methods are divided into detection-based, edge-based, template matching, particle filter, marker-based and depth-based methods. The basic concept of each method is discussed. Section 3 discusses in detail the mathematical concepts needed in implementing a 3D tracking algorithm for a rigid body. These mathematical tools are perspective projection, internal and external camera parameters, pose estimation using linear and non-linear methods, and algorithms for making the tracking process robust. Section 4 presents the main contribution of this thesis: the implementation of the 3D tracking method. Choosing the most suited method for tracking a mobile device under inspection is discussed, and then the implementation of the chosen method is presented in detail. Finally, Section 5 evaluates the performance of the method developed.

2. METHODS FOR 3D RIGID BODY TRACKING

3D object tracking is a problem often encountered in robotics, augmented reality (AR), and computer vision. The goal of 3D object tracking is to estimate the pose of an object using images obtained from a camera. The pose of the object consists of position and orientation relative to the camera. In this thesis, all coordinates and rotations are expressed in Euclidian space. Objects tracked are assumed to be rigid. Therefore the pose consists of three translation coordinates and three rotation angles. The pose is estimated from live camera feed instead of a prerecorded video.

2.1 Detection-based tracking

Detection methods are also suited for tracking applications. Object and its pose are detected from each frame individually. One benefit of tracking by detection is that it is not recursive, contrary to many other tracking methods. Recursive tracking methods are less robust than non-recursive methods because they fail if something goes wrong between consecutive frames. Detection-based trackers are better in recovering from mistakes because every frame is handled individually. Recursive methods also need an initialization, unlike detection trackers.

Objects are detected based on local features found from an image. The local features can be points, regions, or edges visible in the image. One of the most popular methods for feature extraction is scale-invariant feature transformation (SIFT) algorithm [1]. SIFT finds key points from an image. Weighted gradient histograms are calculated and saved for these key points in the training phase. In the detection phase, SIFT features are extracted from a new image and matched with the features saved in the training phase. Nearest neighbor matching is used to find the corresponding key points. SIFT is invariant to image scaling and rotation, and thus the same key points of an object can be found from different images even if the object has been rotated or moved.

Gordon and Lowe use SIFT features for 3D object tracking [2]. In the training phase, a set of images of the object is needed. The training images can be from unknown, spatially separated viewpoints, and they do not need to be pre-calibrated. SIFT keypoints are extracted from the training images, and their 3D positions are recovered by a global optimization over all camera parameters and point coordinates. At detection-phase, features extracted from a frame are matched with the training feature database. From the matched features, a set of corresponding 2D-3D points is acquired. Object pose can be recovered from these corresponding points.

Falsely detected keypoints lead to less stable pose estimation, which causes the jitter effect [3]. Jitter is a problem in particular for detection-based tracking methods because each frame is handled separately, and there is no temporal consistency. A motion model can be used to dynamically smoothen the changes in the pose of the object between frames. Gordon and Lowe minimize jitter by regularizing the solution with the pose from the previous frame [2].

Instead of SIFT features, Hinterstoisser et al. present Natural 3D Markers (N3M) [4]. A set of 4 or 5 feature points that are close to each other and unique geometrically and photometrically can be an N3M. An offline learning stage is needed for extracting N3Ms before they can be used at run time. During the online stage, the feature points are extracted from each frame. A point classifier is used for finding the corresponding N3M points that were learned offline. The pose of the object can then be calculated from these corresponding sets of points.

Detection-based tracking methods estimate the pose from multiple corresponding point pairs. Some algorithm is needed for rejecting the pairs that are mistakenly classified as corresponding. One method for dealing with these errors is the random sample consensus (RANSAC) [5]. Datapoints are divided into inliers and outliers by RANSAC. For example, Gordon and Lowe use RANSAC to make their tracking method robust [2]. RANSAC is discussed in more detail in Section 3.4.

2.2 Edge-based tracking

Edge-based object tracking methods match the edges of a known 3D model with high contrast edges detected from the image. The texture of the 3D model is not utilized by these methods. Edges are naturally stable to changes in lighting, while texture may not be [6]. Edge detection from an image is one of the fundamental operations in computer vision, and there are multiple methods to do it. One of the most popular methods for detecting edges is the Canny detector [7]. It finds edges with image gradients and makes them one pixel wide.

RAPID (Real-time Attitude and Position Determination) is the first real-time edge-based detector [8]. It projects the 3D model into the 2D image plane. The projection depends on the pose estimate from the previous frame. Visible edges of the projected model are divided into control points. The next step is to find the corresponding points for the control points. The corresponding points are searched from a one-dimensional line that is perpendicular to the edge. After finding corresponding points to the control points, a new pose estimate is solved as a least-squares problem. Calculating the pose estimate from a set of corresponding point pairs is discussed in more detail in Section 3.3.

Many improvements have since been proposed to the RAPID tracker, but the basic idea remains the same. The main weakness of the RAPID tracker is its sensitiveness to background and texture clutter, shadows, and partial occlusions. [9] These cause false matches between control points and measurement points. False matches make the new pose estimate less accurate. RAPID does not identify and remove incorrect measurements or use the information about how stable the control points are between frames. Each control point is treated individually, without using the information that the measurements of the points on the same edge are correlated.

Drummond and Cipolla use an M-estimator to make the RAPID tracker more robust [10]. M-estimators are discussed in more detail in Section 3.4. Armstrong and Zisserman divide control points into groups instead of treating each point individually [9]. Each group represents a line, and measurement outliers that do not fit into that line are rejected. Measurements are divided into inliers and outliers using RANSAC, which is a means for improving the robustness of pose estimate, as discussed in detail in Section 3.4 [5]. Armstrong and Zisserman also compute a weight to each line based on how often it is correctly found [9].

Measurement points are found from a one-dimensional search line for each control point. Multiple measurement point candidates can be found from a single search line. Drummond and Cipolla deal with this problem by giving a weight to each control point based on how many candidate measurement points are found [10]. Seo et al. calculate the probability of a measurement point candidate being the correct measurement point for every candidate point [11]. Probabilities are computed comparing histograms of background and foreground with corresponding histograms of measurement candidates.

2.3 Template matching

A template matching tracker compares the content of each camera image with sample templates [12]. The object that will be tracked needs to be well-textured for template tracking to work well. A training stage is required before a template matching tracker can be used in real-time. Templates of the object are extracted during the training stage. Each of the templates contains the pose of the object. Therefore the pose of the object can be estimated by matching template regions with the images from the camera.

Masson et al. divide the surface into small patches [13]. The patches are extracted during an offline learning stage. With multiple patches, only a part of the surface needs to be visible to the camera: the pose estimation is less sensitive to the object being blocked, and the performance of the algorithm is improved.

Jurrie and Dhome estimate an interaction matrix from the textured 3D model during the offline learning stage [14]. The interaction matrix represents the object in such a manner that it can be located from the images obtained from the camera. Multiple interaction matrices are required for tracking full 360° rotations. During the tracking stage, the difference between the frame and the reference pattern is computed. Pose estimation is calculated from corresponding 2D and 3D points.

Ladikos et al. [15] combine feature-based tracking with template-based tracking. Their default tracking method is template-based tracking because it can handle well changes in illumination, image blur, and small movements between frames. Feature-based tracking is used when there are large movements between frames, and the template-based algorithm fails. Due to limited computational resources, feature-based and template-based tracking methods are not concurrent.

2.4 Particle filter

The tracking of a 3D object can be represented as a Bayesian state estimation problem. A particle filter is one of the Bayesian filters, and it estimates the state of the system [16]. The state of the rigid body in a 3D tracking application is the pose of the object: the x-, y-, and z-coordinates and rotations around the x-, y- and z-axis. The state at the time *t* is represented by a symbol $\mathbf{X}_t$. The next state $\mathbf{X}_{t+1}$ can be estimated by adding a small motion to the $\mathbf{X}_t$.

A motion model estimates the next state of the object. A simple motion model can be, for example, the difference between the previous state $\mathbf{X}_{t-1}$ and the current state $\mathbf{X}_t$. The motion model and noise are added to the current state. Noise is added because the motion model is not perfect. In addition to the previous state and the predicted state, a measurement of the current state is needed. Measurement at the time *t* is represented by the symbol $\mathbf{Z}_t$.

A large number of particles are required for satisfactory results. One particle is one state sample of the system. A new cycle starts by first creating a new set of particles from the results of the last cycle. New particles are resampled from the old collection of particles according to probabilities that are the normalized weights of the previous particles. The motion model is applied to all of the new particles to get new predicted states. The next step is to compare the latest measurement data with the updated particles. Particles that match the measurement data better are given bigger weights. The weights are also normalized. The new set of particles and weights are used by the next cycle. The state estimate is the weighted sum of all the particles.

The initialization of the particles is required before the particle filter cycle can start. New states cannot be acquired without the previous states. Choi and Christensen have implemented a 3D object tracking method that uses the particle filter [17]. They have chosen to use chamfer matching for particle initialization [18].

2.5 Marker-based tracking

Markers can be used in 3D tracking if it is possible to modify the object that is being tracked. Markers are visual cues that are easy to detect from an image. The 3D position and orientation of the markers in the body coordinate system are known beforehand. Well designed markers are of high contrast, and they are easy to recognize and identify. The benefit of using markers instead of natural features is that their locations can be measured from the image with higher accuracy. Perspective distortion needs to be taken into account when detecting the markers. E.g., a circular marker can be seen as an ellipse in the image because of the perspective distortion [19].

The markers need to be distinguished from one another if there are multiple markers in use. Colors or unique shapes can be used for marker identification. Black circles with white backgrounds and vice versa are used in augmented reality application made by Hoff and Nguyen [20]. The black and white regions can be detected using a simple thresholding operation because of the high contrast. Instead of black and white, State et al. [21] use colorful markers for more reliable detection. Each marker consists of a central dot and an outer ring, and four colors are used. This configuration enables 12 unique markers for pose estimation.

The optimal size for the marker depends on the distance between the camera and the marker. The tracking system will have a narrow tracking range if all the markers are of the same size. Different size markers have different detection ranges. Cho et al. [22] use multiple size fiducials for expanding the tracking range. The markers are made out of numerous rings, and various colors are used for the rings. The smallest markers have one core circle and one outer ring. As the fiducial size increases, a new ring is added outside of the previous size marker. Fiducials are divided into groups based on how many circles they have.

Only one corresponding point is obtained from each circular marker. Different shapes of the markers can allow for multiple corresponding points from a single marker. Koller et al. [23] use four corners of a square marker as corresponding points between 3D model points and measurement points. There are small red squares inside the marker for identification purposes. The pose is estimated with an Extended Kalman filter once fiducials are found and identified.

Rekimoto [24] uses more complex markers, 2D square-shaped barcodes. Only one marker is needed for pose estimation. The marker contains identification information and acts as a landmark for pose estimation. An image obtained from a camera is first binarized, and then connected black regions are searched for. A heuristic check on the size and aspect ratio is applied to the region candidates bounding rectangles, and the best match is chosen as the marker. The internal and external parameters of the camera are solved by using the four corner points of the marker.

2.6 Depth based tracking

RGB-D cameras have, in addition to a regular RGB camera, a specific type of depth-sensing device that can augment the image with depth information [25]. This depth information can be used by different 3D tracking algorithms to improve performance and accuracy. Kinect is an example of an RGB-D camera that it is widely used for research purposes because it's low cost. Kinect provides depth information for surfaces without texture, unlike depth information received from passive stereo cameras [26].

Park et al. [26] use depth information to improve the template matching tracking method. The training stage is needed for extracting the templates of the object. No 3D model of the object is required in advance. The shape of the object is learned in the training stage. Each template consists of the pose of the object, depth map for the template region, and contour points extracted from the image. At run time, the object is first detected from the color image using the pre-learned templates, and an initial estimate for the pose is acquired. The initial pose estimate usually is accurate for the rotation part, but errors in translation can be large. The original pose estimate is refined with the depth map.

Choi and Christensen [27] use a depth map as part of a particle filter tracker. Their method is the first particle filter method using a depth sensor and able to run in real-time. The design of an efficient and robust likelihood function, which determines the particle weights together with measurement data, is essential for the performance of a particle filter. As measurements for example 3D point coordinates, surface normals, edges from depth discontinuities, and surface texture can be applied. Choi and Christensen define a measurement point using 3D coordinates, normal, and color values so that the measurement points can be compared with the rendering results directly. Direct comparison between measurement points and rendered points allows efficient calculation of likelihood for a large number of particles.

3. MATHEMATICAL TOOLS

Common mathematical expressions in 3D tracking algorithms are introduced in this Chapter. Tracking methods based on a 3D model of the tracked object need a projection model. The 3D model is projected to a 2D image plane. The perspective projection of a 3D model is used in this thesis, and it is discussed in Subchapter 3.1. Internal and external matrices for the perspective projection are discussed in Subchapter 3.2.

Pose tracking methods often find the corresponding points from an image for the projected points. The projection of the model is altered to improve the matching to the measurement points. The location of the projection depends on the pose of the object. The projection matches the measurement if the pose used in the projection matches the pose of the real object.

Pose estimate is calculated from the distances between the projected points and the measurement points. Linear and non-linear methods for finding the optimal pose are discussed in Subchapter 3.3. Finding the measurement points from the image often include errors. Algorithms for dealing with false matches are discussed in Subchapter 3.4. All the coordinates and rotations are expressed in Euclidean space.

3.1 Perspective projection

A pinhole camera model is a simple model for a real-world camera. The pinhole camera model is a pure perspective projection model for projecting points in space onto a plane [28]. The center of the camera is the origin of a Euclidian coordinate system. An image plane is a plane where the z-coordinate is equal to the focal length f of the camera. A point in space is mapped to a point on the image plane where a line joining the point in space and the camera center meets the image plane. Projection of a 3D point $\mathbf{M}_\mathrm{c} = [\mathrm{x}_\mathrm{c},\ \mathrm{y}_\mathrm{c},\ \mathrm{z}_\mathrm{c}]^\mathrm{T}$ to an image plane is shown in Figure 1. Point $\mathbf{m}_\mathrm{c} = [\mathrm{u}_\mathrm{c}, \mathrm{v}_\mathrm{c}]^\mathrm{T}$ is the projection point of the $\mathbf{M}_\mathrm{c}$ to the image plane. Principal point $\mathbf{p} = [\mathrm{u}_0, \mathrm{v}_0]^\mathrm{T}$ is the point where the optic axis of a camera intersects the image plane. C is the location of the camera.

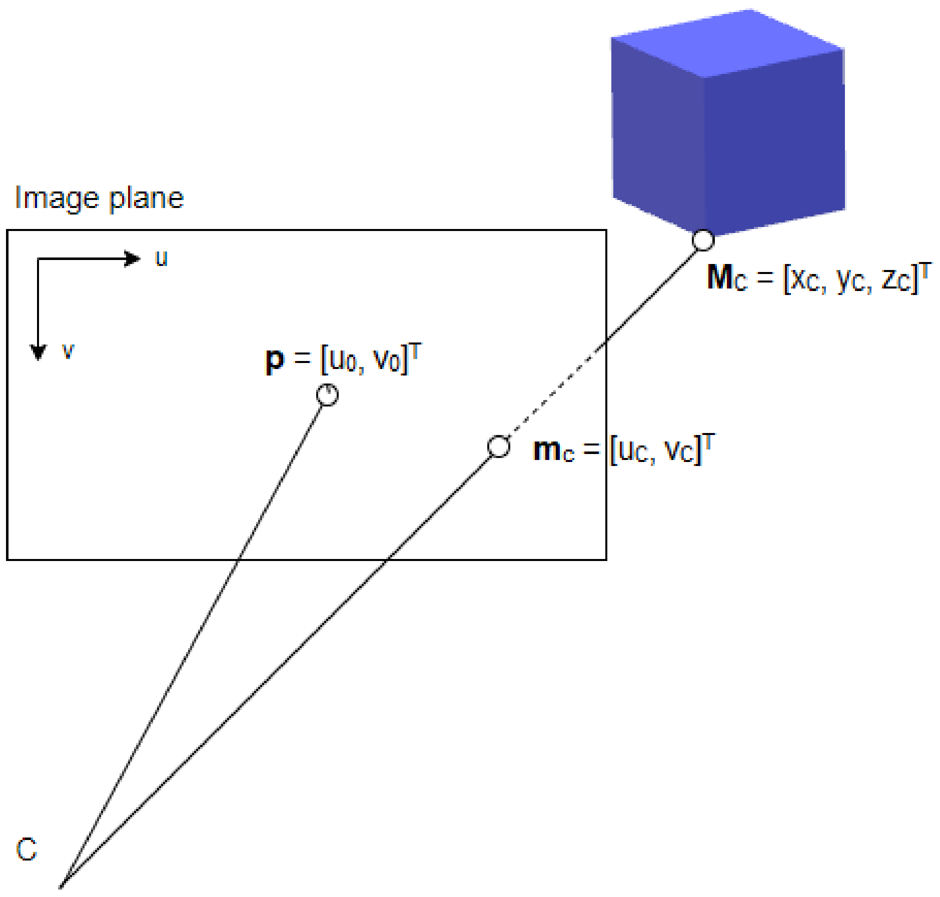

Figure 1. *A 3D point is projected to the image plane.*

The 3D point needs to be converted to a homogeneous coordinate before the projection. Homogeneous 3D point is related to its projection point by

$$s\mathbf{m} = \mathbf{PM}, \tag{1}$$

where s is the scale factor, $\mathbf{m} = [\mathrm{u}, \mathrm{v}, 1]^{\mathrm{T}}$ and $\mathbf{M} = [\mathrm{x}, \mathrm{y}, \mathrm{z}, 1]^{\mathrm{T}}$ are the homogeneous corresponding points, and $\mathbf{P}$ is a 3x4 perspective projection matrix.

3.2 Internal and external matrices

An object is tracked in a coordinate system that is attached to the camera. Coordinates of the object are first transformed from world coordinates to camera coordinates and then projected to the image plane. Projection matrix $\mathbf{P}$ is used to express coordinates of a 3D model in the coordinate system of the camera. The projection matrix is a product of an internal parameter matrix $\mathbf{K}$ and an external parameter matrix $\mathbf{E}$

$$\mathbf{P} = \mathbf{KE}. \tag{2}$$

Internal parameter matrix of the camera is

$$\mathbf{K} = \begin{bmatrix} \alpha_u & s & u_0 & 0 \\ 0 & \alpha_v & v_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}, \tag{3}$$

where α_u and α_v are the scaling factor in u and v direction, $[u_0, v_0]^{\mathrm{T}}$ is the principal point of the image, and s is the skew parameter. The skew is non-zero only if u and v directions are not perpendicular. Matrix $\mathbf{K}$ is also known as a camera calibration matrix, and the camera is calibrated when all the values in $\mathbf{K}$ are known.

In addition to the camera parameters, a matrix that represents the pose of the object is needed. The pose of a 3D object consists of a location and rotation matrices. The external parameter matrix is

$$\mathbf{E} = \begin{bmatrix} \mathbf{R}_{3x3} & \mathbf{t}_{3x1} \\ 0\ 0\ 0 & 1 \end{bmatrix}, \tag{4}$$

where **R** is a 3x3 rotation matrix, and **t** is a 3x1 translation matrix. The rotation matrix is a combination of rotations around x-, y- and z-axis. Rotation around the x-axis is called roll, and matrix representation of it is

$$\mathbf{R}_x(\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{bmatrix}, \tag{5}$$

where α is the rotation angle around the x-axis. Rotation around y-axis is called pitch, and matrix form of it is

$$\mathbf{R}_y(\beta) = \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \tag{6}$$

where β is the rotation angle around the y-axis. Rotation around z-axis is called yaw

$$\mathbf{R}_z(\gamma) = \begin{bmatrix} \cos\gamma & -\sin\gamma & 0 \\ \sin\gamma & \cos\gamma & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7}$$

where γ is the angle of rotation around the z-axis. Rotation matrix **R** is the product of the three rotation matrices

$$\mathbf{R} = \mathbf{R}_z(\gamma)\mathbf{R}_y(\beta)\mathbf{R}_x(\alpha). \tag{8}$$

Each of the rotations $\alpha, \beta,$ and γ can be solved from the rotation matrix. The rotation angles always have more than one solution when they are solved from the rotation matrix [29].

3.3 Pose estimation

Coordinate points of a 3D model are projected to a 2D image plane. The location of the points on the image plane depends on the external parameter matrix $\mathbf{E}$. The projected points need corresponding measurement points. An image obtained from a camera is used for finding the locations of the measurement points.

A set of projected points and a corresponding set of measurement points are used to estimate the optimal value for the pose. In pose estimation algorithms, the difference between the location of the projected points and the measurement points is minimized by adjusting the pose.

3.3.1 Closed-form solutions

Solving the projection matrix $\mathbf{P}$ is necessary for uncalibrated cameras, whereas for calibrated cameras it is enough to solve the external parameter matrix $\mathbf{E}$. Direct Linear Transform (DLT) can be used for solving the projection matrix [30]. The projection matrix consists of 11 free parameters, so if at least six projected points are matched with the corresponding measurement points, a unique solution for the matrix $\mathbf{P}$ exists [6]. Other methods that solve the external parameter matrix instead of the projection matrix require fewer corresponding point pairs because of the $\mathbf{E}$ consists of only six free parameters.

The problem of estimating pose with a given set of n corresponding point pairs is called the Perspective-n-Point (PnP) problem. For 3 corresponding point pairs, the problem is called P3P. Haralick et al. review major direct solutions for the P3P problem and discuss their stability [31]. Quan and Lan introduces an algorithm for 4-, 5-, and n-point pose estimation problem [32].

Pose from orthography and scaling with iterations (POSIT) is an algorithm for solving PnP where n is larger than 3 [33]. The camera model in POSIT is an orthographic projection model so that a linear system can be used for approximating pose. POSIT needs no pose estimate initialization. All the points on the object are assumed to have the same depth. The size variations between the model and the image are due to scaling when the depth remains constant. More than three points on the same plane do not improve the performance of the POSIT algorithm. POSIT has been implemented in the Open Source Computer Vision Library (OpenCV) [34].

3.3.2 Least-Squares Minimization

Least-squares minimization is a popular solution for extrinsic matrix $\mathbf{E}$ estimation from a set of corresponding point pairs. Least-squares is an iterative method, and it can handle noise in the measurements better than the methods above. Least-squares minimizes the distances between the projected points and the measurement points. Least-squares can be represented with a formula

$$\mathbf{E} = \underset{\mathbf{E}}{\arg\min} \sum_{i=0}^{N} \mathrm{dist}^2(\mathbf{PM}_\mathrm{i}, \mathbf{m}_i), \tag{9}$$

where N is the number of corresponding point pairs. The formula minimizes the squared distances between the projected 2D points and their corresponding measured 2D points. Measurement errors are expected to be Gaussian and independent from each other. The least-squares algorithm is initialized with a guess of the matrix $\mathbf{E}$. [6]

Equation (9) can also be written as finding the pose that minimizes the sum of distances between projected and measured points

$$\mathbf{p} = \underset{\boldsymbol{p}}{\arg\min} \sum_{i=0}^{N} \|z(\mathbf{p}) - \mathbf{b}\,\|^2 \tag{10}$$

where $\mathbf{p}$ is the vector containing the pose parameters, $\mathbf{b}$ is the vector containing the measured points, and z is a function that projects the 3D model points into the 2D image plane.

Pose parameter vector $\mathbf{p}$ can be written as the unknowns of a set of linear equations if the function z is linear. In matrix form

$$\mathbf{Ap} \;= \mathbf{b}, \tag{11}$$

and $\mathbf{p}$ can be solved from this equation using the pseudo-inverse of $\mathbf{A}$

$$\mathbf{p} = \left(\mathbf{A}^\mathrm{T}\mathbf{A}\right)^{-\mathbf{1}}\mathbf{A}^\mathbf{T}\mathbf{b}. \tag{12}$$

All the measured points have an equal effect on the pose estimate. Weights can also be given to the measured points. For example, higher weights can be given to higher quality measurements. [6]

A modified method is required if the function z is non-linear. Gauss-Newton and Levenberg-Marquardt are iterative algorithms that can be used for estimating the solution of a non-linear problem. An initial guess is needed for both algorithms, and the pose is updated in every iteration by

$$\mathbf{p}_{i+1} = \mathbf{p}_i + \Delta_i, \tag{13}$$

where Δ_i is a step that is calculated differently based on the method that is being used. The goal is to find the pose that locally minimizes the cost function

$$f(\mathbf{p}) = \frac{1}{2}\big(\mathbf{b} - z(\mathbf{p})\big)^T\big(\mathbf{b} - z(\mathbf{p})\big), \tag{14}$$

where $\mathbf{b} - z(\mathbf{p})$ is the reprojection error. The real cost function is too complicated to be minimized in closed form, and therefore an approximation is used instead. Taylor approximation of the cost function is

$$f(\mathbf{p} + \Delta) = f(\mathbf{p}) + \mathbf{g}^T\Delta + \frac{1}{2}\Delta^{\mathrm{T}}\mathbf{H}\Delta, \tag{15}$$

where $\mathbf{g}$ is the gradient vector $\frac{df}{dp}(\mathbf{p}) = \big(\mathbf{b} - z(\mathbf{p})\big)^T\mathbf{J}$ and H is the Hessian matrix

$$\mathbf{H} = \frac{d^2f}{dp^2}(\mathbf{p}) = \mathbf{J}^{\mathrm{T}}\mathbf{J} + \sum_i (\mathbf{b} - z(\mathbf{p}))^{\mathrm{T}}\frac{d^2z_i}{dp^2}, \tag{16}$$

where $\mathbf{J}$ is the Jacobian matrix of the prediction model. The second derivate of z can be assumed small in comparison to the $\mathbf{J}^{\mathrm{T}}\mathbf{J}$ component of the Hessian matrix if the projection error is small or the model is nearly linear. The Hessian matrix can then be approximated by dropping the second term in formula 16. This approximation is called the Gauss-Newton approximation. [35] The next step is to differentiate f with respect to Δ and set it equal to zero. After rearranging the terms, we get the Gauss-Newton equation

$$\Delta = -\big(\mathbf{J}^{\mathrm{T}}\mathbf{J}\big)^{-1}\mathbf{J}^{\mathrm{T}}\big(\mathbf{b} - z(\mathbf{p})\big). \tag{17}$$

The main advantage of the Gauss-Newton approximation is its simplicity. The second derivates of $z(\mathbf{p})$ are usually very complex and difficult to calculate. The convergence rate of the Gauss-Newton approximation is significantly reduced if the second derivative of the prediction model is not small.

Levenberg-Marquardt (LM) algorithm is similar to the Gauss-Newton algorithm, but it calculates the step Δ slightly differently and ends up with

$$\Delta = -\big(\mathbf{J}^{\mathrm{T}}\mathbf{J} + \lambda\mathbf{I}\big)^{-1}\mathbf{J}^{\mathrm{T}}(\mathbf{b} - z(\mathbf{p})) \tag{18}$$

where the term $\lambda\mathbf{I}$ stabilizes the algorithm. The value of λ is reduced if the new value for Δ reduces the distance between the corresponding points, and the new value for Δ is accepted. The new value for Δ is rejected if it increases the reprojection error. The next iteration is calculated using a larger value for λ. The algorithm stops if there is no value for λ that decreases the reprojection error. With small values for λ the LM algorithm behaves similarly to the Gauss-Newton algorithm. [6]

Both the Gauss-Newton and the LM algorithms can get stuck at a local minimum. The global minimum is more likely to be found with a good initial guess. Computation of the Jacobian matrix is required by both algorithms. Jacobian can be calculated numerically, but an analytical expression is better for speed and accuracy.

3.4 Robust tracking

Tracking methods based on a 3D model project this model into a 2D image plane. Measurement points corresponding to the projected points are needed in pose estimation. In practice, finding the measurement points often include errors. For example, errors can be caused by unfavorable lighting conditions, shadows, occlusion, and clutter in the background or in the texture of the object. Errors in finding the correct measurement points can be substantial. Even a single large error may have a significant impact on the pose estimate determined with least-squares minimization. RANSAC and M-estimators are popular methods to limit how much one corresponding point can affect the pose estimation [5].

RANSAC randomly extracts the smallest possible subset from the available data set required for generating a model. Using the smallest possible subset maximizes the probability that all of the chosen points are accurate measurements, and there are no false measurements among them. A model is generated using the selected points, and all of the data points are divided into inliers and outliers based on how well they fit the generated model. The goal is to find a model that maximizes the number of inliers and minimizes the number of outliers.

RANSAC is an iterative process, and the smallest number of iterations k needed can be calculated by

$$k = \frac{\log(1-p)}{\log(1-w^n)}, \tag{19}$$

where p is the probability that all of the selected points are inliers in at least one of the iteration rounds, w is the probability of choosing an inlier randomly from the dataset, and n is the smallest number of points required for generating a model [5]. After the best division to inliers and outliers is found, the outliers are ignored. A new model is generated using all inlier datapoints.

The least-squares method tries to find model parameters that minimize the squared error between projected and measured points. As the distance is squared, long distances have a massive impact on the estimated parameters. Instead of squaring the distances, other functions can be used. M-estimation minimizes

$$\sum_i \rho(r_i), \tag{20}$$

where r is the distance between the measurement and the predicted value, and ρ is some function. In least-squares regression $\rho(r) = r^2$ and outliers have a large effect on the result because the errors are squared. Least absolute deviation (LAD), or L_1 norm, takes the absolute value of the distance. For LAD the function that is being minimized is $\rho(r) = |r|$. Outliers do not affect the result as much as in least-squares because the is no second power.

Two other popular M-estimator functions are Huber estimator and Tukey estimator. The formula for the Huber estimator is

$$\rho_{Hub}(r) = \begin{cases} \frac{r^2}{2}, & \text{if } |r| \leq c \\ c\left(|r| - \frac{c}{2}\right), & \text{otherwise} \end{cases}, \tag{21}$$

where c is a threshold. Huber estimator is very similar to LAD when $|r| > c$, but unlike LAD, Huber is differentiable when $r = 0$. For Tukey estimator the formula is

$$\rho_{Tuk}(r) = \begin{cases} \frac{c^2}{6}\left(1 - \left(1 - \left(\frac{r}{c}\right)^2\right)^3\right), & \text{if } |r| \leq c \\ \frac{c^2}{6}, & \text{otherwise} \end{cases}, \tag{22}$$

where c is a threshold for Tukey estimator. The Tukey estimator becomes flat when $|r| > c$, which means that large errors between the corresponding points do not have influence in the pose estimate. For small values of r, both the Huber and the Tukey estimators behave similarly to the least-squares estimator. In figure 2, there are least-squares, LDA, Huber, and Tukey estimators drawn in the same figure. The threshold c is set as 1 for the Huber and 3 for the Tukey estimator.

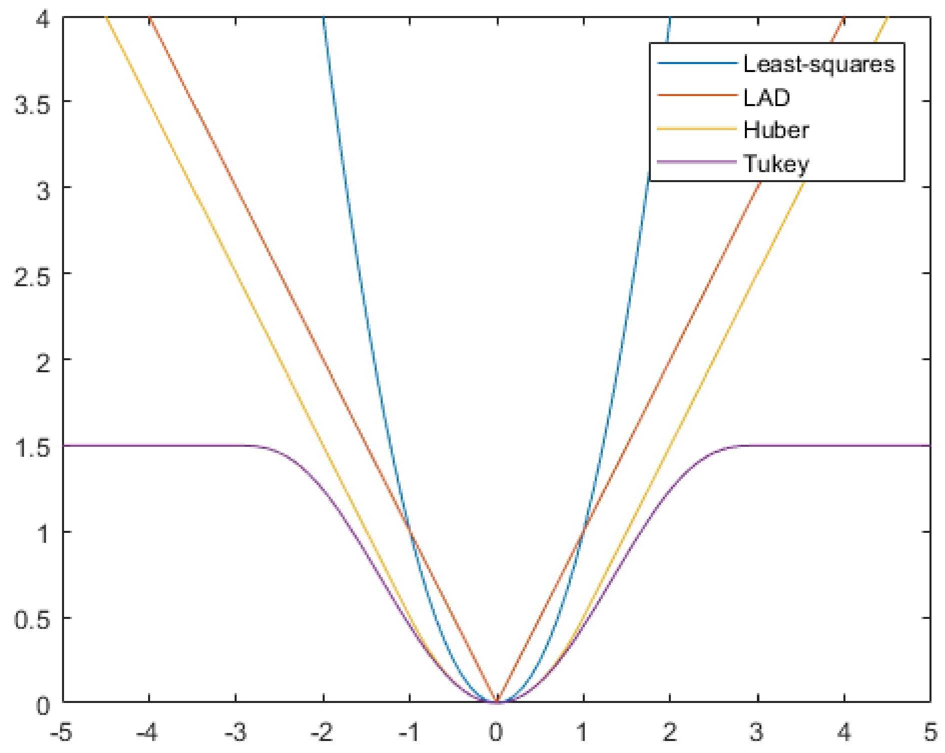

Figure 2. *Least-squares, Least absolute deviation, Huber estimator, and Tukey estimator are drawn in one figure.*

The Huber estimator finds the global minimum more reliably than the Tukey estimator because the Huber is convex, as can be seen from figure 2. The Tukey estimator removes the effect of clear outliers entirely, but it needs to have a good initial guess not to get stuck in a local minimum [6]. For example, the result of RANSAC can be used as an initial guess for the Tukey estimator, but also other options exist.

The Gauss-Newton and Levenberg-Marquardt algorithms can be used with one of the M-estimators. The chosen M-estimator is used to give a weight for each of the corresponding point pairs. The weight w for pair i is

$$w_i = \frac{\sqrt{\rho(r_i)}}{r_i}. \tag{23}$$

A diagonal weight matrix $\mathbf{W} = \mathrm{diag}(\dots\ w_i\ \dots)$ can be formed from all of the weights. The iteration step with weights calculated with the Gauss-Newton method is

$$\Delta = -\left(\mathbf{J}^{\mathrm{T}}\mathbf{W}\mathbf{J}\right)^{-1}\mathbf{J}^{\mathrm{T}}\mathbf{W}\left(\mathbf{b} - z(\mathbf{p})\right). \tag{24}$$

The weight matrix can be similarly added to the Levenberg-Marquardt method. The iteration step of the LM method with the weight matrix is

$$\Delta = -\left(\mathbf{J}^{\mathrm{T}}\mathbf{W}\mathbf{J} + \lambda\mathbf{I}\right)^{-1}\mathbf{J}^{\mathrm{T}}\mathbf{W}\left(\mathbf{b} - z(\mathbf{p})\right). \tag{25}$$

Weights are recalculated in every iteration of the Gauss-Newton and LM algorithms.

4. IMPLEMENTATION

The purpose of the tracking algorithm in this thesis is to keep the identity of a mobile device known while it is turned around by hand. The requirements for the tracking algorithm in this application are discussed in this Section. How well the tracking methods introduced in Section 2 meet these requirements is assessed. The method that is the best fit for this application is chosen and implemented. The steps in the tracking process of the chosen method are discussed in detail.

4.1 Tracking environment and performance requirements

Before the object can be tracked, a mobile device is detected and identified from a camera stream. Identification information is displayed on the screen of a mobile device using a QR code. Tracking can be started after a QR code containing correct identification is read. The device is turned around by hand, so the backside becomes visible to the camera. A picture is taken of the backside of the device. The number of scratches on the back cover is detected from the picture. This thesis focuses only on the tracking stage. The device is not identified, and no picture of the backside is taken while evaluating the performance of the implemented tracker.

During the tracking, both the camera and the object that is being tracked may move. The device needs to be identified again if it leaves and re-enters the field of view of the camera. The tracking algorithm is not required to recover identification if the object leaves and re-enters the field of view of the camera.

The tracking algorithm must run on an Android mobile device as an Android application on most of the high-end Android devices. Most Android devices have a camera that can be used for tracking. Mobile devices have limited computational capability compared to desktop computers, which is why one requirement for the tracking algorithm is limited computational complexity. Low frame rate and tracking failure may result from too high computational complexity.

The objective of the pose estimation is to keep the identity of a mobile device known while it is being turned by hand. The tracker is required to be able to track different mobile devices. Small errors in the pose estimate are not critical for performing this objective. The background of the device during the tracking is not defined, and it can be cluttered. The tracking algorithm must be able to perform in different environments. During the tracking, the mobile device will be rotated by hand. The hand will block part of the device, so it is not acceptable for the tracker to fail if a small part of the object is not visible to the camera. The object can be rotated around any axis, so the tracker is required to cope with any rotation.

4.2 Choosing the tracking method

The non-recursive nature of detection-based tracking methods gives them an advantage over the recursive algorithms. They can recover from errors between consecutive frames because detection is done individually for each frame. The previous pose estimate is not used in the calculation of a new pose estimate. The detection-based methods can recover if the object leaves and re-enters the field of view of the camera without reinitialization, unlike recursive methods. This feature would be useful in this application because there is no guarantee that the object will stay in the area that is visible to the camera, but it is not required.

A significant downside to the detection-based methods is that they depend on the texture of the object. In this application, it is essential to be able to track any mobile device based on the shape only. The texture of the object is unknown because it varies between different mobile devices. The detection-based trackers also require a training stage before they can be used in real-time. The application must be able to track many different device types. Tracking of different device types would require retraining the detection-based tracker each time a new device type becomes to be inspected. For these reasons, a detection-based tracker is not the appropriate choice for this application.

Marker-based tracking methods can be accurate and fast. They modify the environment to assist in the tracking process. Markers would be easy to add to the screen of the mobile device that will be tracked. These markers could then be used for accurate tracking. Markers on the backside of the device would be required for tracking the device while only the backside is visible to the camera. It is important to be able to track both sides of the mobile device in this application. Adding markers to the backside of each device that will be tracked would not be practical for the use case in this thesis. For this reason, a marker-based tracker was rejected.

Tracking algorithms using the particle filter method have yielded accurate results. Choi and Christensen have been able to track objects that are held in hand and are partially blocked from the view of the camera [17]. High accuracy and reliability are achieved by having a large number of particles. More particles mean more computational load.

Karlsson et al. have analyzed the complexity of the particle filter by counting the number of required floating-point operations [36]. A particle filter tracking method is a viable choice if computational complexity is not a problem. In this application, the tracker is running on a mobile device. Computation capability, limited on mobile platforms, excludes the particle filter from this application.

The edge-based trackers are lighter to run than particle filter trackers, and thus are more suitable for mobile devices. The recent edge-based methods can be made robust against background and foreground clutter, which was a big problem in the first generation of the edge-based trackers. The weakness of the edge-based methods is their recursiveness, which makes them vulnerable to drift. An edge-based tracker does not require a training stage. The tracking is based on the edges of the object, and the texture is not used. The same tracker can be used for tracking different mobile devices. The edge-based tracking method was found as the most suitable tracking method for the application in this thesis.

4.3 Projecting a 3D model and finding the outermost edges

A 3D model of the object is required by the edge-based tracking method. There are different methods for acquiring the model. For example, Izadi et al. use the depth sensor of a moving Kinect camera to create a 3D model in real-time [37]. The object that is being tracked in this thesis is a mobile device. The same 3D model is used for tracking various devices because mobile phones are similar in shape. A different 3D model is required for tracking differently shaped devices, such as tablets. The 3D model need not match perfectly the real device.

In this thesis, the 3D model was drawn with a computer-aided drawing program. The model provides only the shape, not the texture of the device. This is because the surface texture varies between devices, and thus cannot be used in the tracking process. For example, the location of the rear-facing camera is not a suitable anchor point for tracking because the location of it varies between devices.

The model file contains the 3D coordinates for all the vertices of the model. The file also includes a list of vertex pair indices that are connected by an edge. The vertex points can be projected to the image plane with Equation (1). The intrinsic parameters of the camera in matrix $\mathbf{K}$ need to be known before the points can be projected. Elements of $\mathbf{K}$ are shown in Equation (3). In this application, the values for α_u and α_v are 4000. Skew s is set to 0, and the principal point is [540, 960]. In Figure 3 the 3D model is projected using pose $\mathbf{p} = [0,\ 0,\ 50,\ \frac{\pi}{4},\ \pi,\ \frac{5\pi}{8}]$.

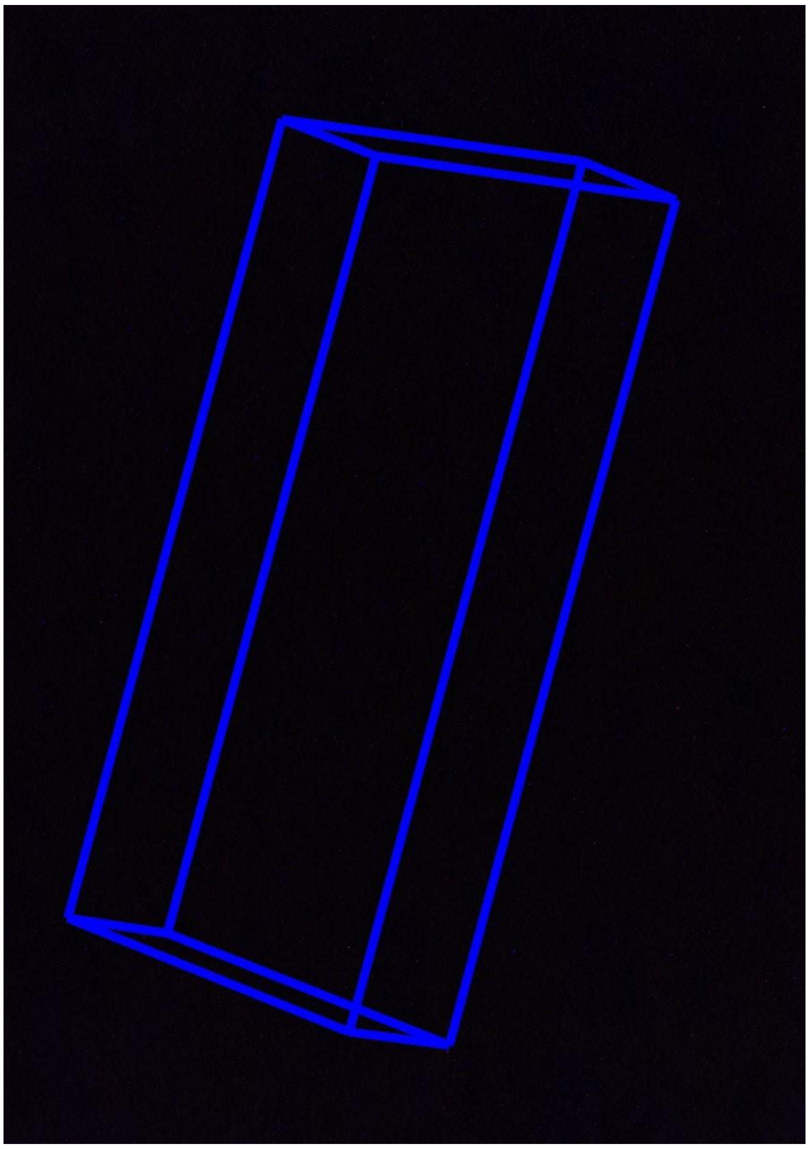

Figure 3. *A 3D model is projected to an image plane.*

The projected edges of the model are divided into inner and outer edges. To do this, the first step is to find the bottom-right vertex by choosing the vertex with the largest y-coordinate. If there are multiple vertices with the same y-coordinate, the one with the largest x-coordinate is chosen. The next step is to find all the vertices that are connected to the bottom-right vertex by an edge. The smallest angle between a connected edge and the x-axis needs to be found because it is one of the outermost edges. The vertex at the other end of the chosen edge is added to a list of outermost vertices.

All vertices connected by an edge to the most recently found outermost vertex are found next. The angles between the connected edges and the latest outermost edge are calculated. The angle is calculated in the clockwise direction. The edge with the smallest angle is one of the outermost edges, and the vertex at the other end of it is added to the list of outermost vertices. This process is repeated until all the outermost vertices are found, which is detected by the bottom-right starting vertex being the next candidate for a new outermost vertex. The algorithm for finding the outermost vertices is described in Program 1.

```
Input: edges, a list of all edges,
       vAll, a list of all vertex points
Output: vOutermost, a list of outermost vertices
vStart = findBottomRightVertex(vAll)
vCurrent = null
vPrev = null
vOutermost = null
while (vCurrent != vStart):
      if (vCurrent == null)
            vCurrent = vStart
      endIf;
      vConnected = findConnectedVertices(edges, vCurrent)
      smallestAngle = 2*Pi
      for (vNext in vConnected):
            if vPrev == null:
                  angle = calculateAngleBetweenXAxisAndEdge(vCurrent,
                        vNext)
            else:
                  angle = calculateAngleBetweenEdges(vPrev, vCurrent,
                  vNext)
            endIf;
            if (angle < smallestAngle):
                  smallestAngle = angle
                  vNextOuter = vNext
            endIf;
      endFor;
      vPrev = vCurrent
      vCurrent = vNextOuter
      vOutermost.add(vNextOuter)
endWhile;
return vOutermost;
```

Program 1. *Algorithm for finding the outermost edges.*

The result of this algorithm can be seen in Figure 4. The outermost edges are drawn in red color in the figure. All of the inner edges are drawn in blue. The 3D model, the camera parameters, and the pose are the same in Figures 3 and 4. The outermost vertices need to be filtered from all of the vertices every time the pose is updated.

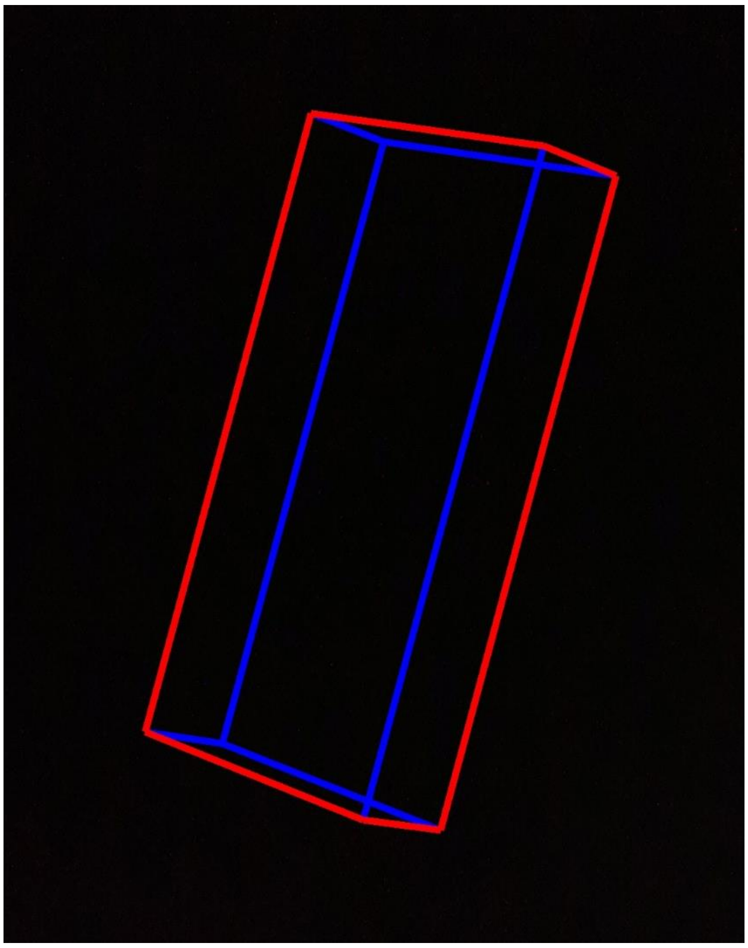

Figure 4. *Outermost edges are found from a projected 3D model.*

The tracking algorithm implemented in this thesis only uses the outermost edges. The inner edges of the projected model are discarded. An image obtained from a camera is used for finding the corresponding edges for the projected edges.

4.4 Finding corresponding points

The edges of the real object corresponding to the projected edges need to be found from an image. To do this, the projected outermost edges are divided into control points. The corresponding points to the control points that are found from the image are called measurement points. One-dimensional search lines $\boldsymbol{l}_i$ are used for finding the measurement points. It is much simpler to look for the measurement points from a search line than from the whole image. Each control point has its own search line. Search lines are perpendicular to the outermost edges of the projected 3D model. The control points are always located in the middle of the search lines. In Figure 5, the outermost edges have been divided into the control points. A search line is drawn for each of the control points in green.

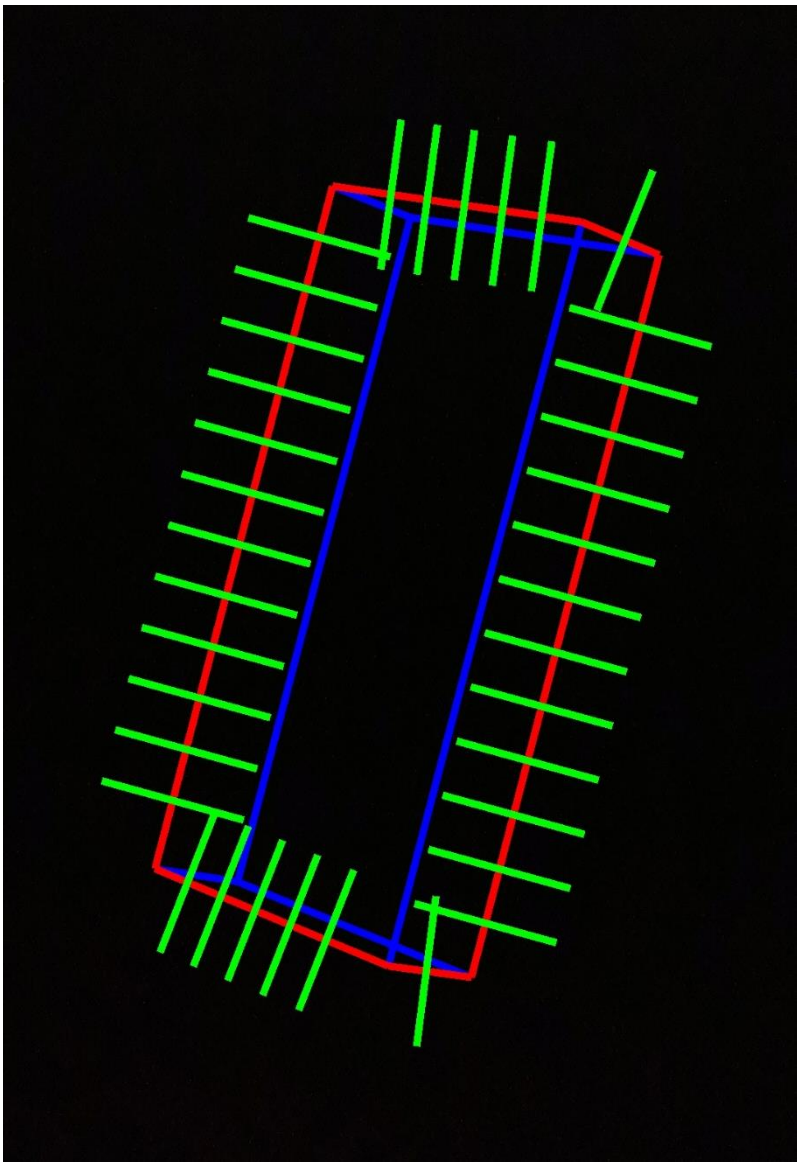

Figure 5. *Search lines are drawn for each of the control points.*

The search lines are extracted from the image, rotated, and stacked on top of each other to create a searching bundle $\mathbf{L}$. The measurement points should match the control points. When the points match, the foreground of the object is located on the left side of the searching bundle. The background is located on the right side of the searching bundle. Each horizontal line of the searching bundle represents one of the search lines. One measurement point needs to be found from each of the search lines.

The resolution of the searching bundle is the width of the search lines multiplied by the number of search lines. The resolution of the image is much greater than the resolution of the searching bundle. Images with a smaller resolution are faster to process. The pixel values of the green search lines are extracted, rotated, and stacked on top of each other to create a searching bundle in figure 6. The original image and the search lines are located at the bottom, and the searching bundle can be seen at the top of Figure 6.

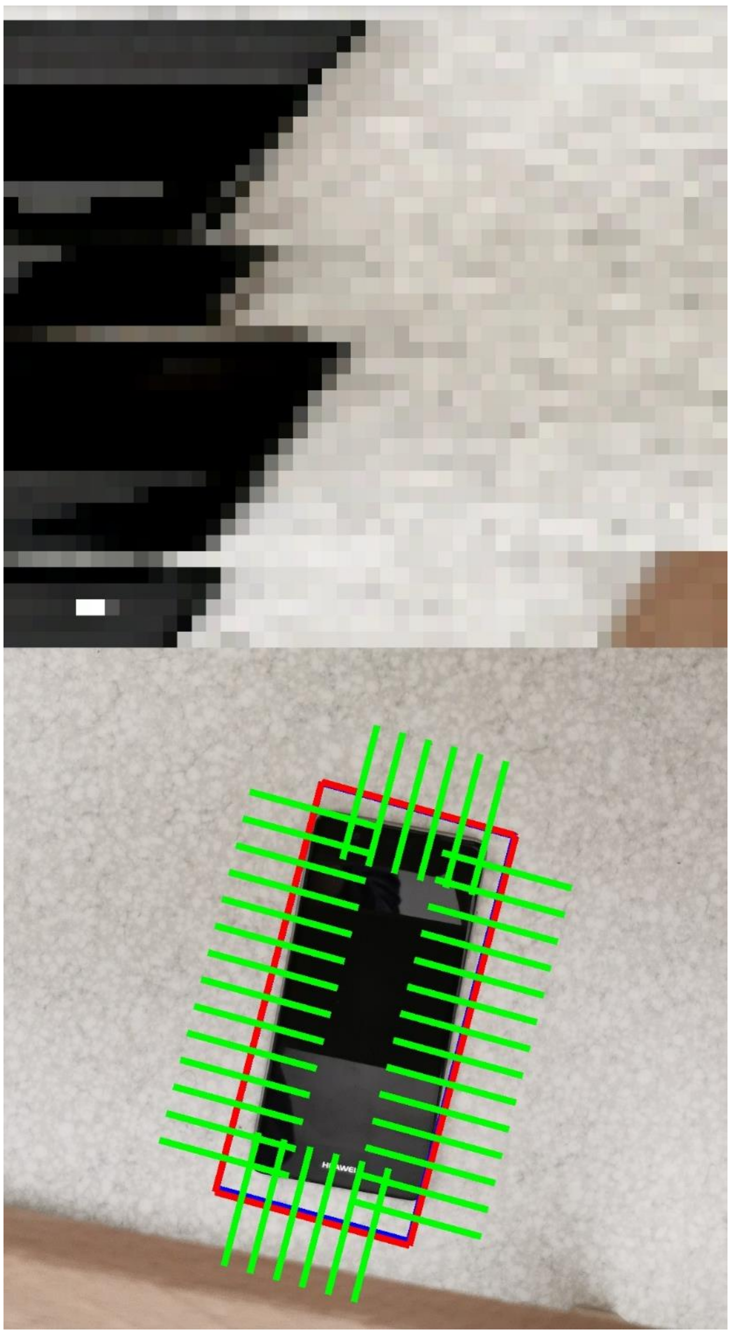

Figure 6. *A searching bundle and the original image with green search lines.*

At the bottom of Figure 6 is an image of a dark mobile phone against a light background. The 3D model is projected on top of the image in red and blue. The pose used in the projection does not match the real pose of the mobile phone because the edges of the model do not match the edges of the phone. Projected edges are divided into control points, and a search line is drawn for each of the control points using green color. The pixel values of the search lines are extracted from the image and used for creating the searching bundle. The searching bundle is located in the top half of the image.

The searching bundle is used for finding the measurement points that correspond to the control points. Each horizontal line of the searching bundle is one of the searching lines. Control points are located at the middle point of each horizontal line in the search bundle. One measurement point can be found from each of the horizontal lines. Measurement point candidates $\boldsymbol{c}_i$ are found by doing 1-dimensional convolution for each row. In this thesis, the search bundle is converted from RGB to hue, saturation, and value (HSV) color space. Only the value channel is convolved because it yielded the best results when testing the performance with combinations of the RGB and HSV channels.

Filter mask [-1, -1, 0, 1, 1] is used in the convolution. The gradient response for the value channel is calculated for all possible points on the search bundle

$$c_{ij} = \|\nabla I_v(l_{ij})\|, \tag{26}$$

where l_{ij} is a location on the search line $\boldsymbol{l}_i$, and $I_v(.)$ is the intensity of the value channel. The resulting c_{ij} is added to the candidate point list $\mathbf{c}_\mathbf{i}$ if it is larger than some threshold value. The threshold is set to 0.6 in this thesis.

The candidate points are drawn with magenta on top of the searching bundle at the top of Figure 7. Each search line has multiple candidate points. The candidate points are used for finding edges from the image. All the edges are successfully detected by the candidate points. Several candidate points are not the edges of the phone; firstly reflections on the screen of the mobile phone are identified as candidate points; secondly shadows are often recognized as candidate points; and thirdly edge on the bottom right of the searching bundle that is not part of the mobile device is found as candidate points.

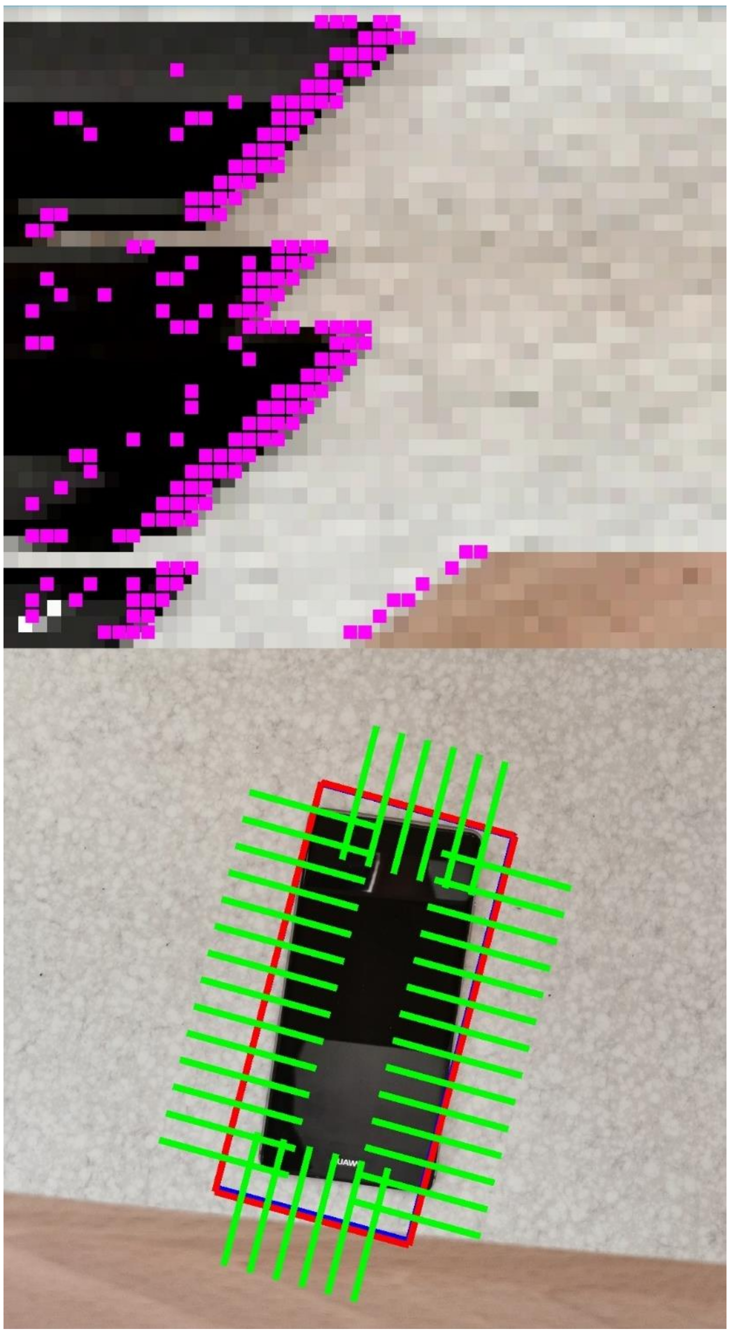

Figure 7. *Candidate points are drawn with magenta on top of the search bundle.*

Multiple candidate points are produced by a single edge. The real edges of the mobile phone are thicker than one pixel in the image, so the convolution value is large at several consecutive pixels. In Figure 7, on each search line, a response to a single edge of the phone can be seen as 2-4 magenta candidate points next to each other.

The multiple candidate points responding to a single edge can be combined to only one point using one-dimensional non-maximum suppression. Only the largest value of a neighborhood is saved. The convolution value of the candidate point is compared to the convolution value of the neighboring candidate points. All the candidate points that are not the largest one in their neighborhood are discarded. In this thesis, 3 neighbors are used in the non-maximum suppression step. The result of the non-maximum suppression can be seen in Figure 8.

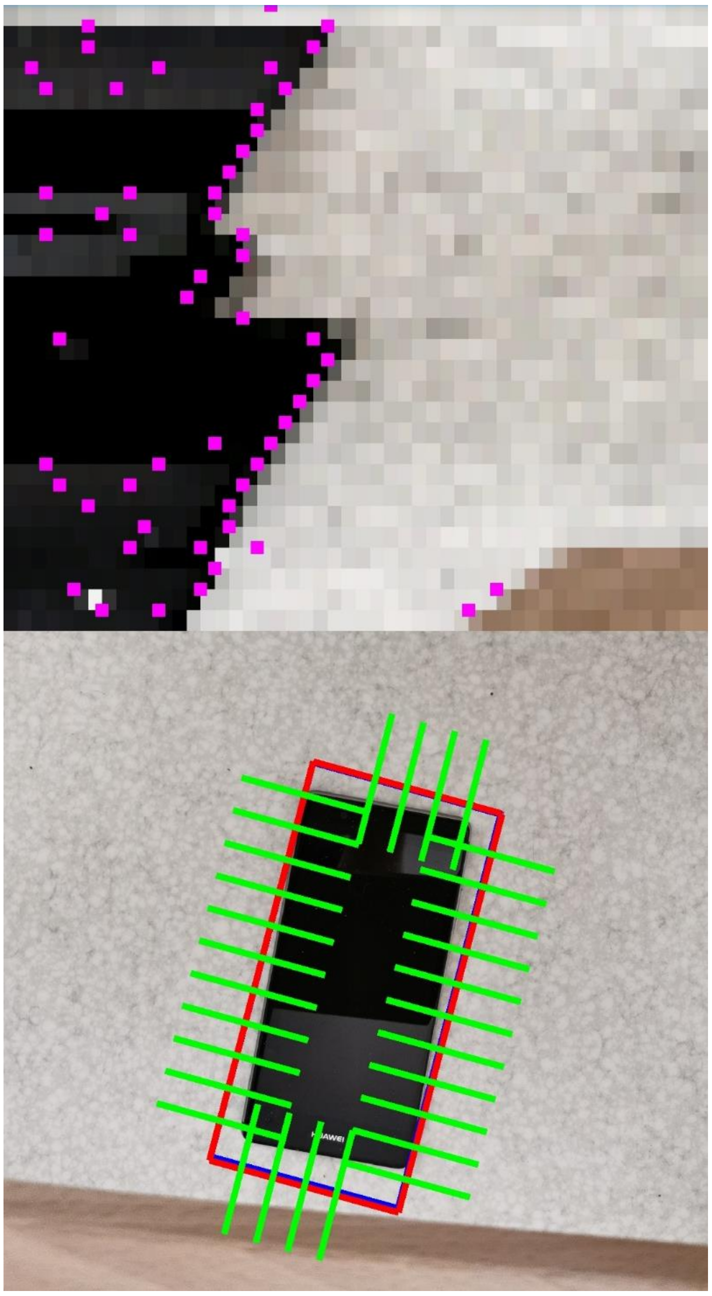

Figure 8. *Non-maximum suppression is applied to the result of the convolution step.*

There can be only one measurement point corresponding to one control point. The best point among the candidate points needs to be chosen if there are more than one candidate points on a search line. The probability that the candidate point is the correct measurement point is calculated for each candidate point. The probabilities of the candidate points belonging to the same search line add up to 1. No measurement point for a search line is found if there are no candidate points on that search line.

The probability of a candidate point being the measurement point is calculated using histograms. Histograms of the background and foreground regions are calculated from the search bundle $\mathbf{L}$. The pixel values of the background area are added to the background histogram, and the same is done to the foreground pixels. The pixels that are on the left side of the control points in the search bundle belong to the foreground area, and the right side of the control points belongs to the background area.

Similar to Peréz et al., HSV values of the pixels are saved in the histograms instead of the RGB values [38]. The HSV color space is used because it is less sensitive to illumination changes [39]. The V component of HSV is the most sensitive to illumination changes, so it is not included in the histogram. HS histogram that is composed of H histogram with N_H bins and S histogram with N_S bins has a bin number of $N = N_H N_S$. The kernel density of the histogram is

$$H(\Omega) = \{h_n(\Omega)\}_{n = 1,\ldots,N}\,, \tag{27}$$

where $h_n(\Omega)$ is the probability of bin n inside an area Ω. The histogram for the foreground region is $H^f(\Omega^+)$ and for the background region is $H^b(\Omega^-)$. The area Ω^+ is the left column of the searching bundle and the area Ω^- is the right column of the searching bundle.

Background and foreground histograms need to be formed for all of the candidate points. Foreground histogram for a candidate point c_{ij} is $H^f(\Phi_{ij}^+)$ and background histogram of the point is $H^b(\Phi_{ij}^-)$. The pixels that belong to the Φ_{ij}^+ are on the left side of the candidate point on the search line $\boldsymbol{l}_i$. The pixels on the search line that are on the right side of the candidate point c_{ij} belong to the Φ_{ij}^-. The areas $\Omega^+, \Omega^-, \Phi_{ij}^+$ and Φ_{ij}^- are drawn in Figure 9.

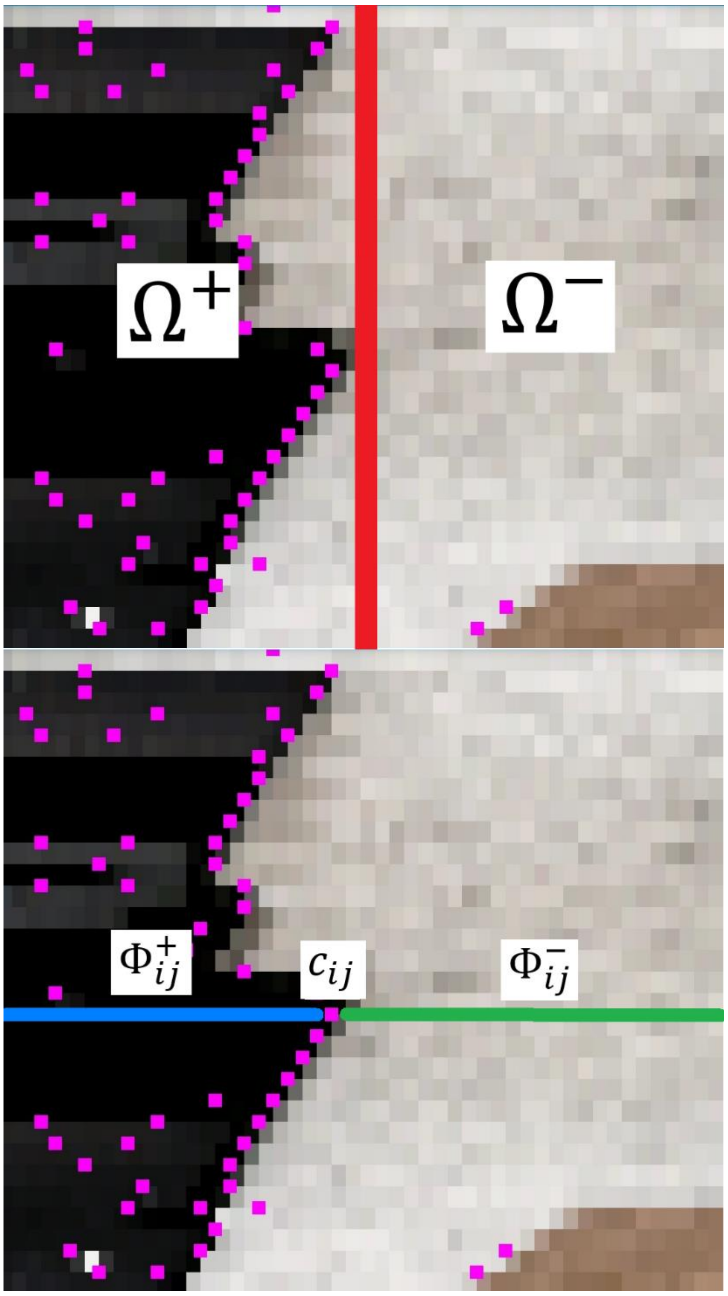

Figure 9. *Background and foreground regions.*

At the top of Figure 9 is a searching bundle divided into two columns with a red line. The pixels that are on the left side column belong to the foreground region Ω^+, and the pixels on the right side belong to the background region Ω^-. At the bottom image is the same search bundle with foreground and background regions drawn for a single candidate point. Candidate point c_{ij} is drawn with a magenta square, the foreground area of Φ_{ij}^+ with blue color and the background area Φ_{ij}^- with green.

The foreground histogram of the candidate points is compared to the foreground histogram of the whole search bundle. The same comparison is made to the foreground histograms. The Bhattacharyya coefficient is a measure of the similarity between two histograms [40]. The coefficient between the foreground histogram of the search bundle and the foreground histogram of a candidate point c_{ij} is

$$BC_{ij}^{f}[\Omega^{+},\Phi^{+}] = \sum_{n=1}^{N} \sqrt{h_n(\Omega^{+}) h_n(\Phi_{ij}^{+})}, \tag{28}$$

and, similarly, the coefficient for the background histogram of the search bundle and the candidate point can be calculated by

$$BC_{ij}^{b}[\Omega^{-},\Phi^{-}] = \sum_{n=1}^{N} \sqrt{h_n(\Omega^{-}) h_n(\Phi_{ij}^{-})}. \tag{29}$$

For the candidate point c_{ij} that is the real measurement point m_i of the search line, both the Bhattacharyya coefficients BC_{ij}^{f} and BC_{ij}^{b} are large. At least either of the BC_{ij}^{f} and BC_{ij}^{b} is small if the candidate point c_{ij} is not the measurement point m_i. Using the Bhattacharyya coefficients, a probability that the candidate point is the measurement point can be defined as

$$p(c_{ij}|m_i) = \frac{1}{Z} BC_{ij}^{f} BC_{ij}^{b}, \tag{30}$$

where Z is the normalizing constant that ensures $\sum_j p(c_{ij}|m_i) = 1$.

Candidates that have low BC_{ij}^{f} or BC_{ij}^{b} value are filtered out with a threshold. The threshold is set to 20 in this thesis. The probability of the candidate point being the measurement points is set to 0 if one of the coefficients is below the threshold value. Weak responses are filtered out because doing so ensures that the object that is being tracked can be found using the search lines. No measurement points should be found if the edges of the object are not on the search lines.

Probabilities are calculated for all the candidate points on a search line, and the candidate point with the highest probability is chosen as the measurement point. The same process is done for all the search lines. The measurement points are drawn in a different color than the other candidate points in Figure 10.

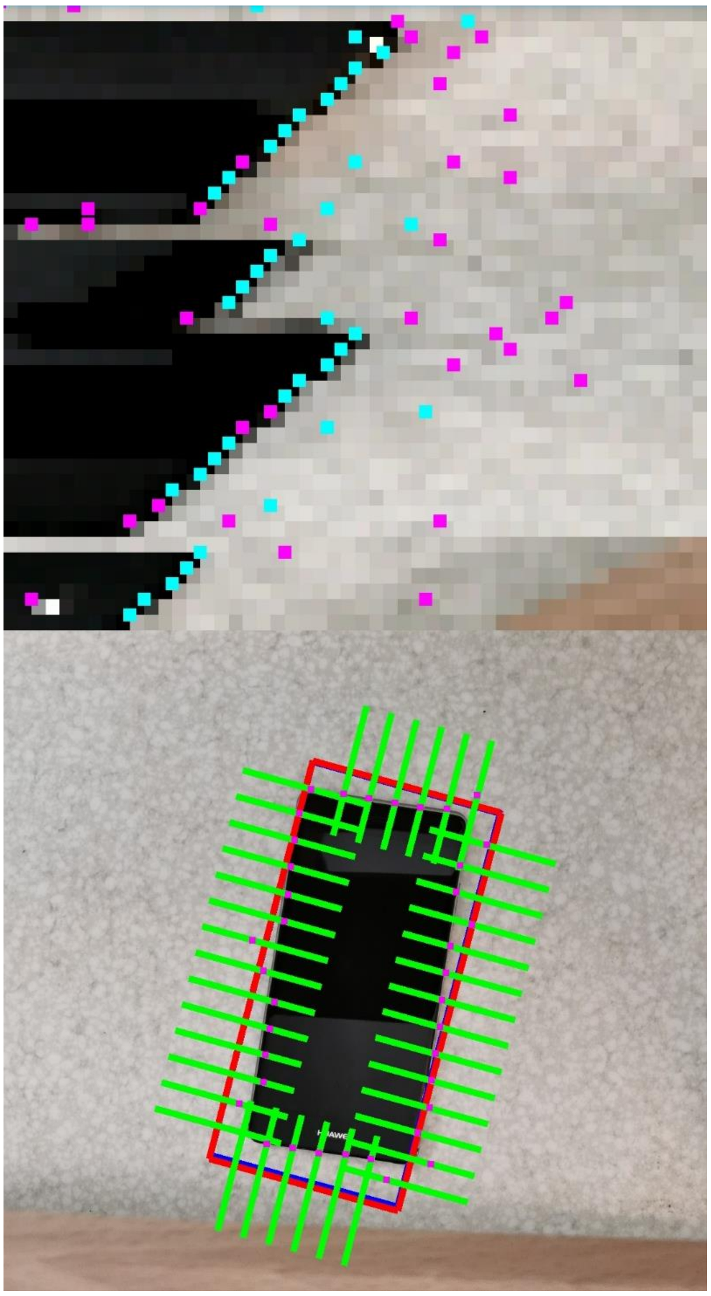

Figure 10. *The measurement points are found from among the candidate points.*

At the top of Figure 10 is the searching bundle, and at the bottom is the original image with the projected model drawn on top of it. The candidate points are drawn using magenta, and the measurement points are drawn with cyan on top of the search bundle. Most of the measurement points are found correctly. The measurement points should be located on the edge where the black left side meets the grey right side. Some cyan points are on the background area because they are not detected successfully. None of the measurement points are in the foreground area of the mobile phone. At the bottom of the figure, the measurement points are drawn on top of the green search lines using magenta. The correct measurement points are located on the edge of the mobile phone.

A new pose estimate is calculated using the control points and the measurement points. The value for the pose does not need to perfect. It is enough that the new estimate is better than the current one. For this reason, it is enough that most of the measurement points are found correctly.

4.5 Estimating pose

The pose is adjusted based on the distances between the control points and the measurement points, found as described above, by applying the methods discussed in Section 3.3. Closed-form algorithms, such as DLT and POSIT, can be used for solving optimal pose from a corresponding set of points. The main benefit of using these methods is that they are fast to execute, and they do not require a lot of computational resources. The main weakness of these algorithms is that they do not cope well with errors in the set of measurement points. There is no guarantee that the solved pose is close to the real pose if the set of measurement points contains errors. Not all measurement points are found perfectly by the implemented method, as can be seen in Figure 10. Closed-form solutions are not viable in this application because the measurement points are not found without errors.

Iterative minimization algorithms perform better than closed-form algorithms if there are errors in the measurements. The iterative nature of these algorithms makes them slower to execute than non-iterative algorithms. Finding the measurement points without errors in all conditions is not realistic for the use case in this thesis. Therefore, an iterative algorithm, either the Gauss-Newton or the Levenberg-Marquardt, is the best choice for finding the optimal pose. The Gauss-Newton algorithm does not require the calculation of the stabilization term, which makes it slightly less complex and easier to implement. This thesis uses the Gauss-Newton algorithm for calculating a new estimate for the pose.

The iteration step of the Gauss-Newton algorithm is calculated using Equation (17). A set of control points and a set of corresponding measurement points are needed before the formula can be used. In addition to the point sets, an image Jacobian $\mathbf{J}$ is required by Equation (17). The image Jacobian is a combination of Jacobians for all the corresponding point pairs.

The first step when calculating the Jacobian for a generic homogeneous point $[\mathrm{X}, \mathrm{Y}, \mathrm{Z}, 1]^{\mathrm{T}}$ is transforming it into the camera coordinates. Coordinates of the point in the camera coordinate system are

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \mathbf{P} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \tag{31}$$

where **P** is the same projection as in Equation (2). Function g transforms the camera coordinates to a point on the image plane

$$g(\mathbf{p}) = \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} \frac{X_c}{Z_c} \\ \frac{Y_c}{Z_c} \end{bmatrix}, \tag{32}$$

where **p** is a vector containing the pose parameters. Jacobian for the point can be calculated from the function g by partial derivative

$$\mathbf{J}_p(\mathbf{p}) = \left[\frac{\partial g(\mathbf{p})}{\partial \mathbf{p}}\right] = \begin{bmatrix} \frac{\partial x}{\partial t_x} & \frac{\partial x}{\partial t_y} & \frac{\partial x}{\partial t_z} & \frac{\partial x}{\partial r_x} & \frac{\partial x}{\partial r_y} & \frac{\partial x}{\partial r_z} \\ \frac{\partial y}{\partial t_x} & \frac{\partial y}{\partial t_y} & \frac{\partial y}{\partial t_z} & \frac{\partial y}{\partial r_x} & \frac{\partial y}{\partial r_y} & \frac{\partial y}{\partial r_z} \end{bmatrix}, \tag{33}$$

where t_x is the translation along the x-axis, t_y is the translation along the y-axis, t_z is the translation along the z-axis, r_x is the rotation around the x-axis, r_y is the rotation around the y-axis and r_z is the rotation around the z-axis. The Jacobian needs to be calculated for each control point that has a corresponding measurement point. The point Jacobians can then be combined into the image Jacobian by placing them on top of each other

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_{\mathrm{p1}} \\ \mathbf{J}_{\mathrm{p2}} \\ \vdots \\ \mathbf{J}_{\mathrm{N}} \end{bmatrix}, \tag{34}$$

where N is the number of control points that have a corresponding measurement point. The dimension of the image jacobian $\mathbf{J}$ is $2\mathrm{N} \times 6$.

The pose is updated by adding the iteration step to the old pose as in Equation (13). The 3D model is projected to the same image using the updated value for the pose. New measurement points are found for the new projection. The distances between the control points and measurement points are calculated using the new points. Tukey estimator with the threshold set to 15, as discussed in Section 3.4, is used for giving weights for all of the measurement points. The calculated pose is the final pose if the error between the control points and measurement points is small enough. The new iteration step is calculated if the distance between the point pairs is not small enough, and the maximum number of iterations for one frame is not reached. The algorithm used to find the optimal pose is presented in Program 2.

```
Input: image, an image obtained from a camera,
       pose, the previous pose,
       3DModel, the model of the object
Output: pose, a new estimate for the pose
for (iterationCount = 0; iterationCount < maxIterationRounds;
++iterationCount):
      controlPoints = GetControlPoints(pose, 3DModel)
      for (controlPoint in controlPoints):
            measurementPoint  =  GetMeasurementPoint(controlPoint,  im
            age)
            //Check that measurement point is found for this control
            point
            if (measurementPoint.isValid()):
                  measurementPoints.append(measurementPoint)
            else:
                  controlPoints.remove(controlPoint)
            endIf;
      endFor;
      //Check   that   enough   control   points   have   corresponding
      measurement      points
      if (measurementPoints.amount() < pointAmountThreshold):
            break;
      endIf;
      //If the distance between the control points and measurement
      points is small enough pose estimate is good enough
      reprojectionError = CalculateError(controlPoints,
                                         measumentPoints)
      if (reprojectionError < maximumAllowedError) {
            break;
      }
      pose   =   CalculateNewPose(controlPoints,   measurementPoints,
      pose);
endFor;
return pose;
```

Program 2. *Algorithm for finding the optimal pose.*

The algorithm is executed when the previous execution of the algorithm is done, and a new frame is obtained. The latest pose, the 3D model, and the latest image are given as the input for the algorithm. The pose and the 3D model are used for obtaining all of the 2D control points. The next step is to find the measurement points corresponding to the control points. A control point is discarded if no corresponding measurement point is found for it. The pose estimate is not updated if too many control points are without measurement points, and the algorithm returns the same pose that was given to it as an input.

After enough measurement points are found, the distances between the measurement points and the corresponding control points are calculated. The pose estimate is accurate enough if all of the distances between the measurement and control points are smaller than a threshold value. The threshold is set to 5 pixels in both the x- and the y-direction in this thesis. A new pose estimate is calculated if there is at least one distance between a measurement and a control point that is larger than the threshold. The image Jacobian is calculated using all of the control points that have a corresponding measurement point. The next iteration of the pose is calculated with Equation (17). A pose estimate is iterated at most six times. After the six iterations, the algorithm returns the most recent pose estimate. The algorithm needs to be executed with a new image if there is not good enough pose estimate found within the six iteration steps.

5. PERFORMANCE EVALUATION OF THE TRACKER

The main objective of the implemented tracker is to be able to track a mobile device while it is being turned by hand. The environment where the tracking is being done is not defined. The tracker must work regardless of the background. The movement speed of the device affects the performance of the tracker. The contrast between the color of the device and the background affects the performance because it needs to be large enough for finding the measurement points reliably. This section discusses the factors that have an effect on the performance of the tracker.

The tracker created in this thesis is an Android application, and it can be installed on different Android devices. The performance of the tracker is dependent on the device that it is installed on. Some devices have more computational capacity than others. The algorithm in Program 2 is faster to execute on the more advanced devices. The faster execution of the algorithm results in a higher rate of processed frames.

The movement of the object is smaller between frames because the object has less time to move. The tracking is less likely to fail if the movements between processed frames are small. The tracking fails if the object moves out of the range of the search lines. A high frame rate does not affect how well measurement points are found or how accurately a new pose estimate is found from a single image.

The tracking application is tested on three different Android phones for this thesis, a Huawei P30 Pro, an OnePlus 5, and a Huawei Mate 10 Lite. During the tracking, the frame rate is 7-12 fps (frames per second) for the P30 Pro, 4-7 fps for the OnePlus 5, and 2-6 fps for the Mate 10 Lite.

5.1 Color of the device and the background

The color difference between the object and the background is used for finding the measurement points. It is easier to find the device correctly when the background is of a different color than the device. The convolution in Equation (26) gives larger values if the contrast is large. Calculating the probabilities with Equation (30) is also more reliable with a large contrast. In Figure 11, a black mobile phone is tracked against a white background.

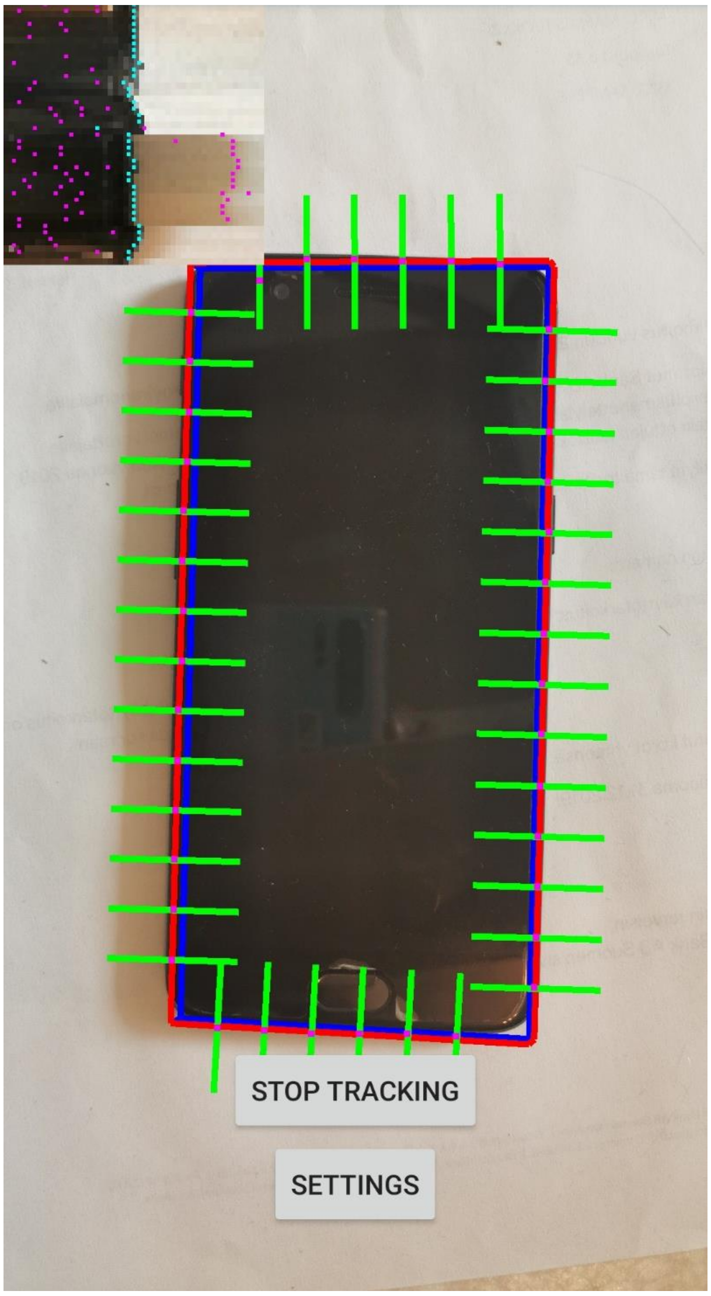

Figure 11. *A black mobile phone being tracked against a white background.*

In the middle of Figure 11, there is a black mobile phone that is being tracked. The background in the image is white. On top of the image, the outermost edges of a 3D model are drawn in red. The search lines for the control points are drawn using green color. The search bundle made of the search lines is located in the top-left corner of the Figure. The candidate points are drawn using magenta, and the measurement points are drawn in cyan on top of the search bundle. There are also two buttons on the bottom of the image, one for stopping the tracking and one for opening a settings menu.

The tracking is successful in the image because the edges of the 3D model match the edges of the real mobile phone. The cyan measurement points form almost a straight vertical line on top of the searching bundle. It is easy to find the correct measurement points because of the color difference between the object and the background. The magenta points on the searching bundle are the candidate points that were not chosen as the measurement point. They are located mainly on the left side of the searching bundle because there are reflections on the screen of the phone. The candidate points on the right side of the searching bundle are caused by the shadow of the phone. Figure 12 is an example of an environment that is more difficult for the tracker.

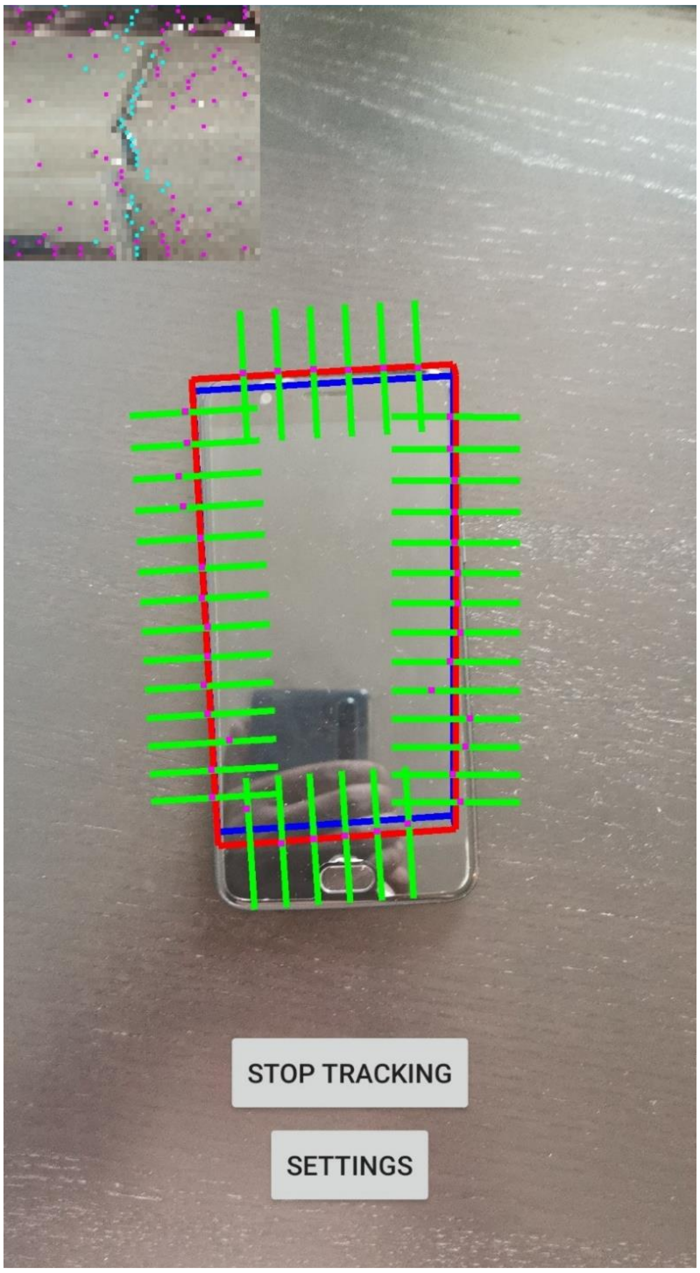

Figure 12. *A black mobile phone being tracked against a black background.*

The layout in Figure 12 is the same as in Figure 11. In the middle of the figure is a black mobile phone against a black background. Using a pose estimate obtained from the tracker, a 3D model is projected on top of the phone. Search lines in green color are drawn perpendicular to the edges of the 3D model. The search bundle obtained by combining the search lines is drawn in the top-left corner. Candidate points are drawn in magenta, and measurement points are drawn in cyan on top of the searching bundle.

The tracker has failed to correctly detect the pose of the phone from the image. Because the pose is incorrect, the bottom edge of the mobile phone does not match the bottom edge of the 3D model. This is caused by the lack of contrast between the object and the background. Unlike in Figure 11, the cyan measurement points do not form a straight vertical line in the searching bundle in Figure 12. There are multiple errors in finding the correct measurement points. Regardless of these errors, the left, right, and top edges are detected pretty well by the tracker. The least-squares minimization discussed in Section 4.4 makes it possible not to completely lose track of the object in this kind of situation. Closed-form methods for pose estimation discussed in Section 3.3.2. would not be as accurate in this scenario because of the measurement errors. Another situation where one of the edges is not detected correctly is shown in Figure 13.

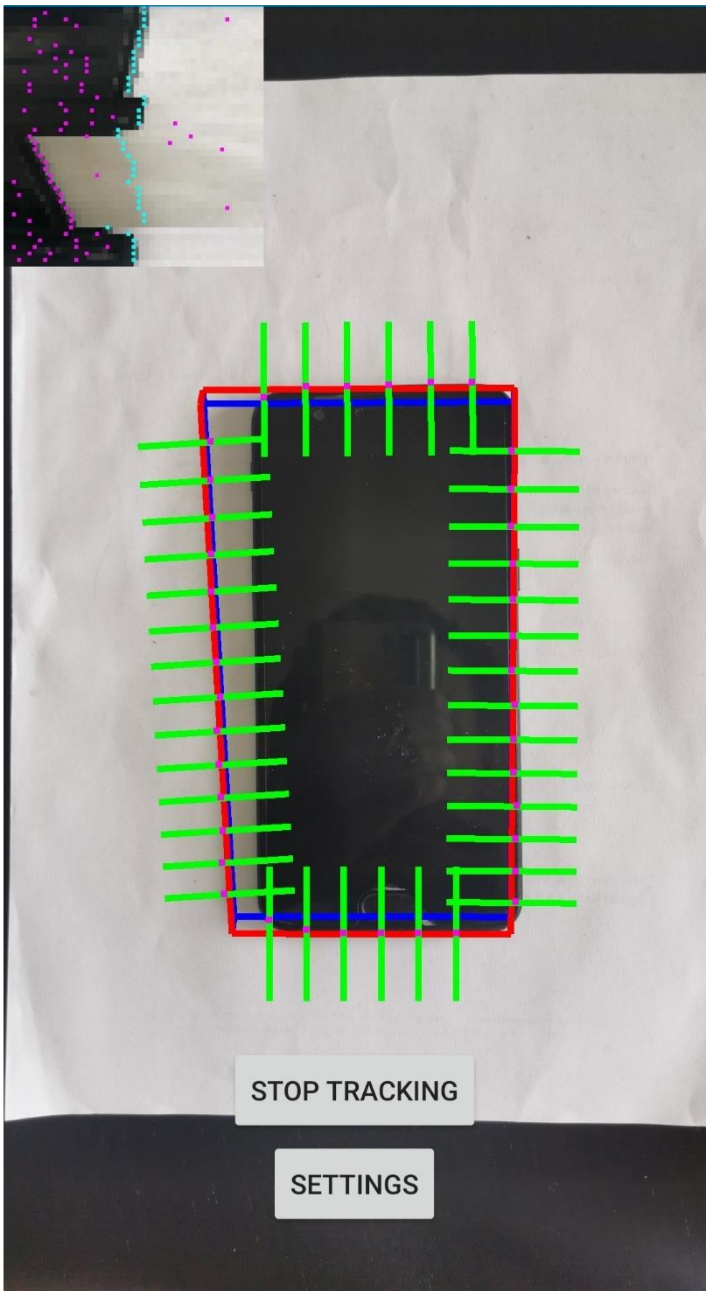

Figure 13. *The shadow of the phone is detected as an edge.*

In Figure 13 is a black mobile phone against a white background being tracked. The layout is the same as in Figures 11 and 12. There is a 3D model, searching lines, and a searching bundle with candidate and measurement points drawn on top of the original image. The left edge of the model does not match the edge of the phone. The other three edges of the model match much better the real edges of the phone.

The left edge of the model is detected incorrectly because of the shadow of the mobile phone. The shadow forms an edge that is parallel to the left edge of the phone. The tracker can mistakenly detect edges in the background area as the edges of the object if they are parallel. The mistakes are more common if the background area between the false edge and the real edge is the same color as the phone. The color affects the result because of the coefficients calculated in Equations (28) and (29). In Figure 13, the false edge is located in the background area, but there can also be clutter leading to false detections in the foreground area. In Figure 14, the bottom edge is incorrectly detected because of an edge in the foreground area.

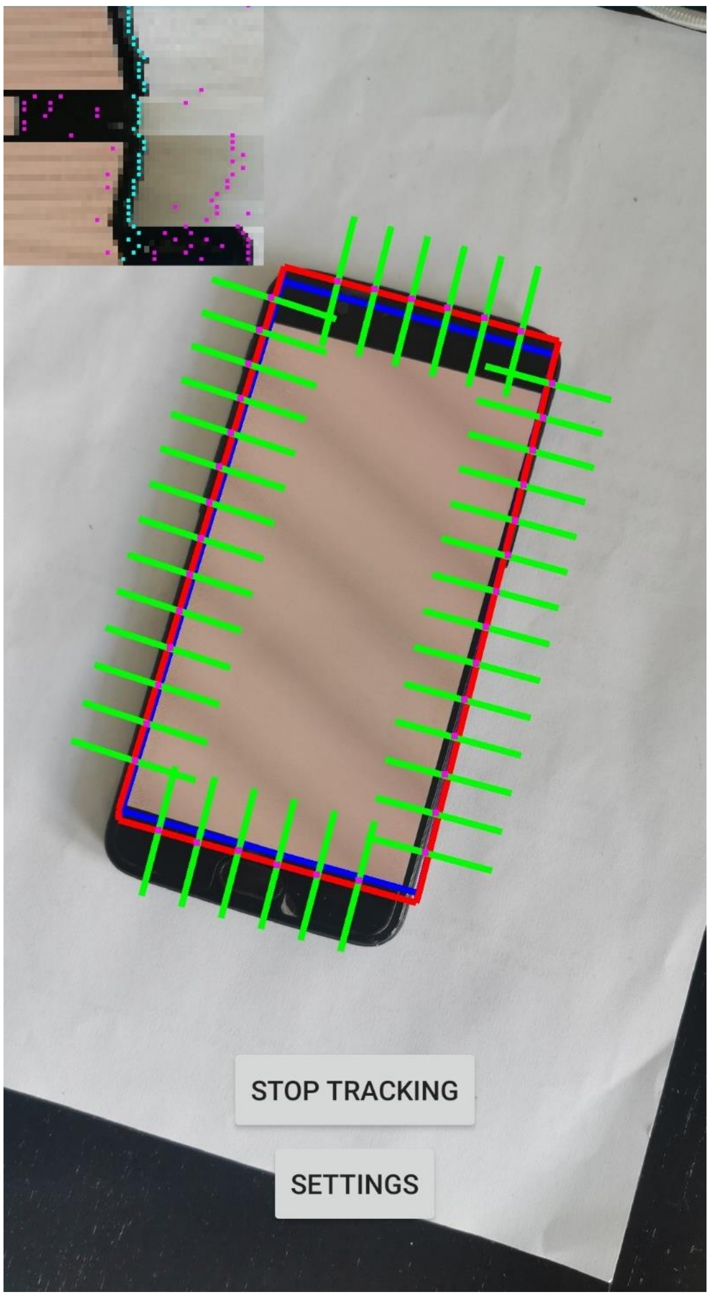

Figure 14. *The bottom edge is incorrectly detected because of the white screen.*

The layout in Figure 14 is the same as in the previous figures. Black mobile phone is being tracked against a white background. The screen of the phone is on, and a white image is shown on the screen, contrary to the previous figures with the screen off. The screen creates edges to the foreground area if the screen is of a different color than the body of the mobile phone. The edges of the screen can be detected as the edges of the phone and cause inaccurate pose estimation. The bottom edge of the phone in Figure 14 is not detected correctly because of the edge of the screen.

5.2 Partial occlusion

In an optimal scenario, the entire mobile phone is visible to the camera at all times. This is rarely the case for the use case in this thesis because the phone is rotated by hand. It is difficult to grab and rotate the phone without blocking at least part of it. For this reason, the tracker must be robust against partial occlusion. Depending on how the phone is grabbed, a different portion of the device is blocked. A minor portion of occlusion does not affect the performance of the implemented tracker. The tracking is successful in Figure 15 despite a thumb blocking a part of the phone.

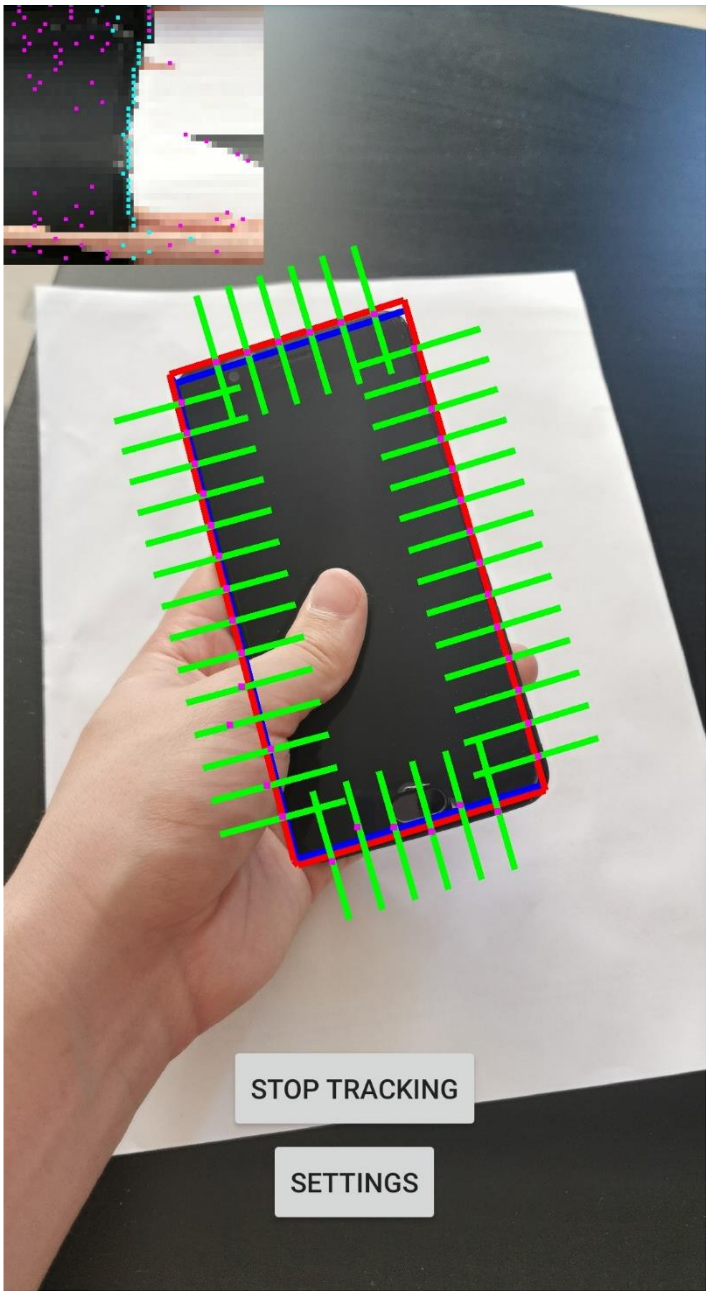

Figure 15. *Part of the mobile phone is blocked by a thumb.*

Black mobile phone is held in hand and tracked against a white background in Figure 15. The 3D model is drawn on top of the image using the estimated pose. Search lines are drawn on top of the model, and the search bundle is located in the top left corner. Candidate and measurement points are drawn on top of the searching bundle. The thumb of the hand holding the device blocks part of the left edge of the phone. The pose estimate is accurate in the figure regardless of the thumb.

Hand is a problem for the tracker because it blocks part of the edge of the phone. It is impossible to find the correct measurement points if they are not visible to the camera. The cyan measurement points are not found correctly in the bottom part of the search bundle. There are enough measurement points visible to the camera so that the tracking is successful. In Figure 16, too many measurement points are blocked, which causes the tracking to fail.

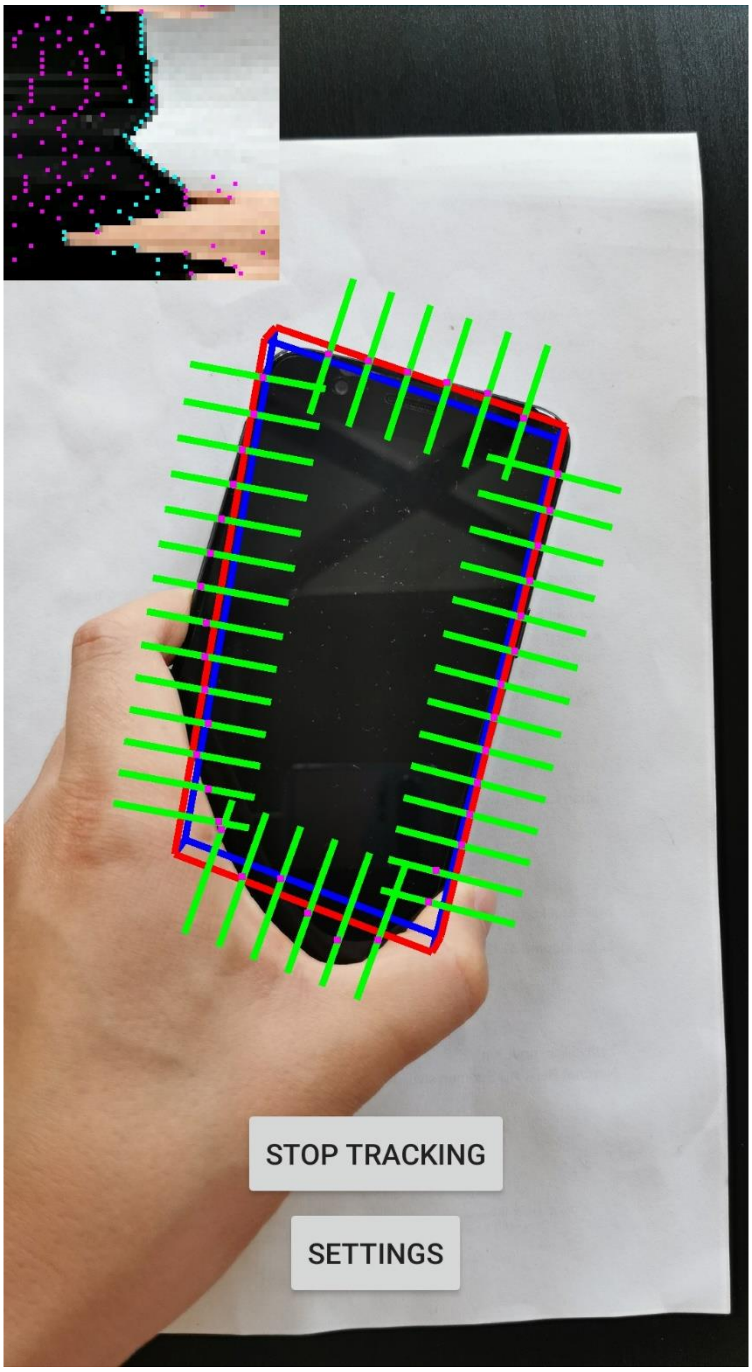

Figure 16. *Tracking fails because a hand blocks a too large area of the phone.*

Black mobile phone is held in hand and tracked in Figure 16. The hand blocks the whole bottom edge of the phone. The estimated pose is used for projecting the 3D model of the phone on top of the image. The edges of the model do not match the edges of the phone, which means that the pose estimate is not accurate. The inaccuracy is caused by the hand blocking a large area of the phone. There is one cyan point on each horizontal line on the search bundle in the top left corner, so measurement points are found for each of the control points. The found points do not match the real points, so a model can not be fit well to the points. For this reason, the cyan measurement points on top of the search bundle do not form a vertical line.

5.3 Tracking a rotating mobile phone

The main objective of the tracker is to keep track of a mobile phone while it is being turned by hand. How well the implemented tracker performs in this task depends on several factors. One of these factors is the calculation capacity of the phone on which the tracker is running. How fast a single frame is processed depends on the calculation capacity. A tracker running on a low calculation capacity device cannot keep up with objects that move fast. A lower calculation capacity is enough only if the object moves slowly.

The color of the background affects how easy it is to find the correct measurement points. The tracker performs better if the background is of a different color than the object. The tracker looks for edges of the mobile phone from the image. Other edges that are parallel and close to the edges of the phone may cause the tracking to fail. Good contrast between the object and the background and no other edges near the edges of the phone increase chances of successful tracking. Successful tracking of a phone being rotated by hand is shown in six stages in Figure 17.

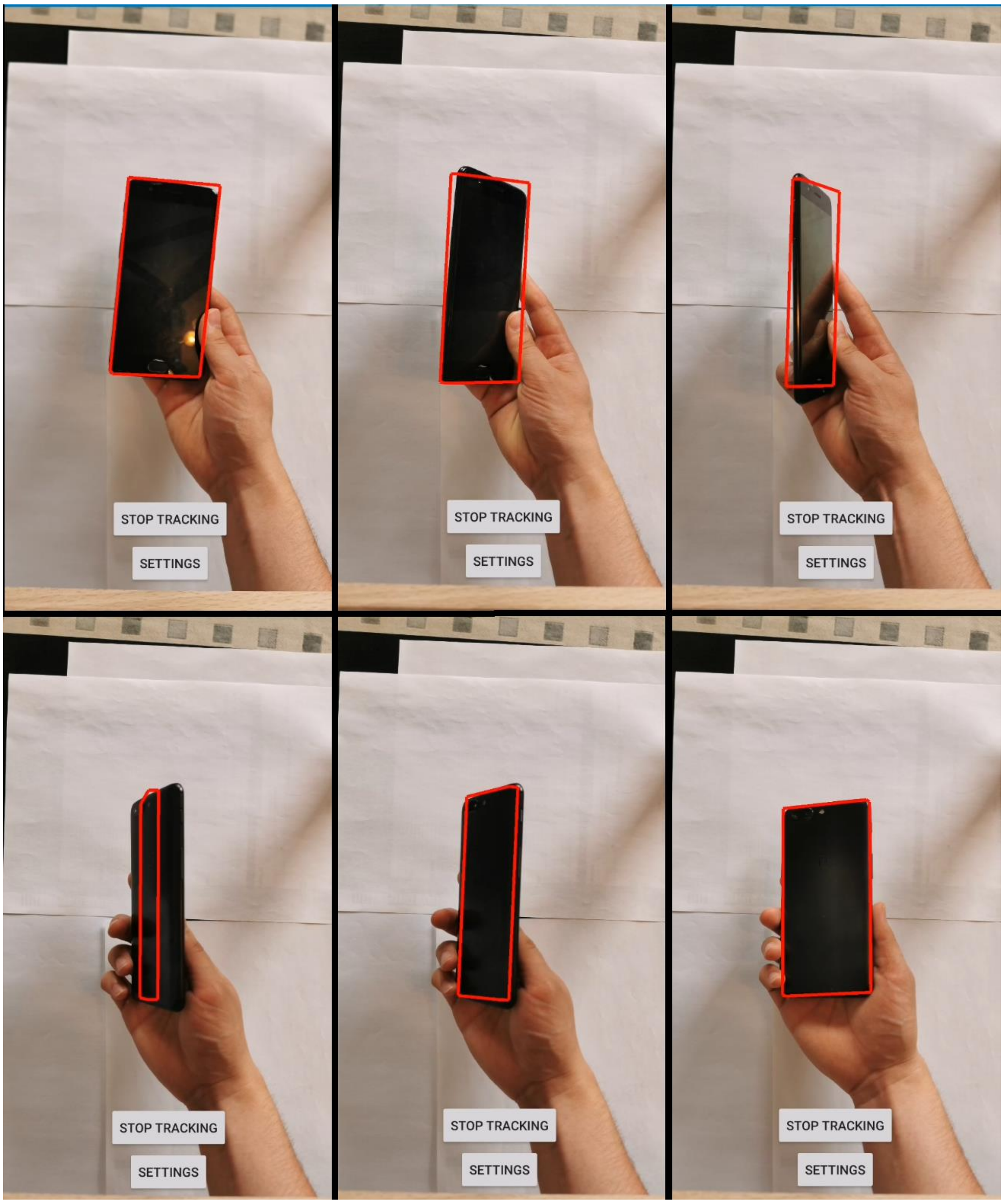

Figure 17. *Successful tracking of a mobile phone while it is turned by hand. Tracker is running on Huawei P30 Pro.*

Figure 17 consists of six images that are in chronological order starting from the top left and ending in the bottom right one. Tracker is running on Huawei P30 Pro. In the images, a black mobile phone is tracked while it is turned by hand. The outermost edges of the 3D model are drawn on top of the images in red. The red edges match the edges of the phone in all of the images except the bottom left one. The tracker follows the object with a small delay caused by the execution time of the tracking algorithm in Program 2. The computational capacity of the phone that the tracker runs on determines the length of the delay. An example of a failing tracking result is shown in Figure 18.

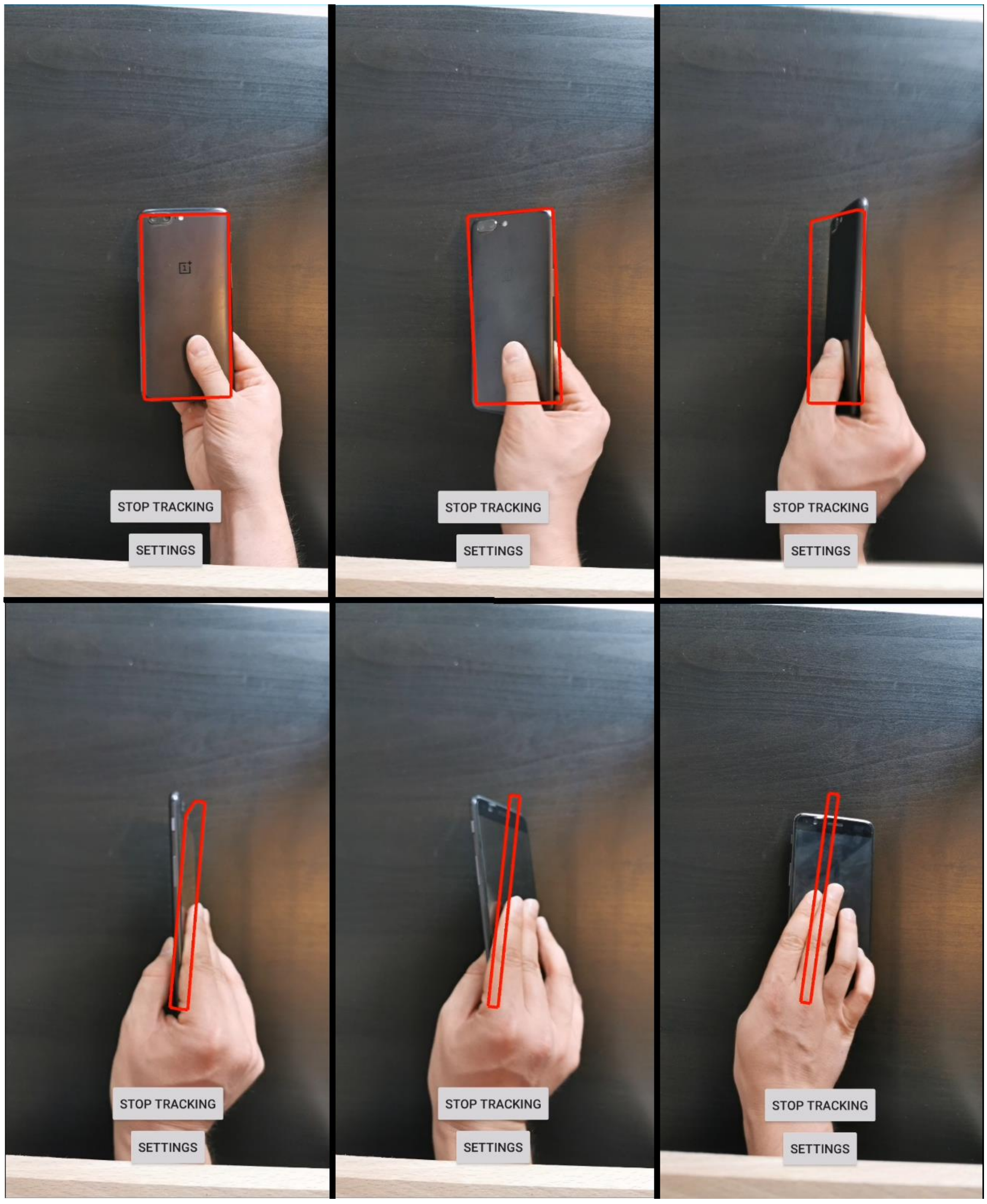

Figure 18. *The tracking of the rotating phone fails. Tracker is running on Huawei P30 Pro.*

A tracking sequence is divided into six images in Figure 18. Similar to Figure 17, the tracking result is drawn on top of each image in red. The tracking is not successful because the red edges do not match the edges of the phone in the bottom half of the figure. The step between the top right and the bottom left images is crucial to the success of the tracking because only a small part of the phone is visible to the camera. The left and right edges of the phone are close to each other, so the tracker can confuse them with one another. The background is the same color as the phone, which increases the chances that the tracking will fail. The phone is gripped in a manner that makes a large area of the phone not visible to the camera. A combination of these factors makes the tracking fail in Figure 18.

Too fast movement speed of the tracked phone can cause the tracking to fail. The phone is moved by hand, so measurement of the movement speed is difficult. The exact slowest speed that causes failure depends on the device that the tracker is running on. The maximum allowed movement speed would be required to be measured for different devices, and it is not done in this thesis. The movement speed is the only factor dependent on the device that the tracker is running on. For this reason, the tracking of a rotating phone is demonstrated only on Huawei P30 Pro.

6. CONCLUSION

In this thesis, methods for tracking the position and orientation of a rigid 3D object were researched. The tracking method should be able to track a mobile device while it is being turned around by hand. How well each of the different tracking methods would fit this application was estimated based on their merits and shortcomings. An edge-based tracking method was chosen as the most suitable because it does not require a training stage. Other advantages of the edge-based method are that the same model can be used for tracking different device types, and the algorithm is light enough for being executed on an Android phone.

The mathematical tools that are needed in the implementation of the tracker are discussed in detail. The first step in the tracking process is to project the 3D model of the object to the image plane. The next step is to find the outermost edges of the projected model and divide them into control points. Each of the control points should have a corresponding measurement point. The measurement points are looked for from the image using a search bundle that is created by combining search lines. A probability that a measurement point is the correct one is calculated using histograms. A new estimate for the pose is calculated with the Gauss-Newton algorithm using the two sets of corresponding points, control points and measurement points.

The implemented tracker can perform the task that it was supposed to, but it has limitations. The object that is being tracked cannot move too much between consecutive processed frames. How much movement is allowed depends on the device that the tracker is running on. Tracking may also fail if the background is the same color as the object. Parallel edges to the edges of the object may also cause tracking failure. A grip of the object that makes a large area of the object not visible to the camera can also cause the tracking to fail.

The most critical part of the tracking is when the phone is on its side, and only a small part of the phone is visible to the camera. A motion model that predicts the future movement of the object based on the previous pose estimates could be added to make the tracking more reliable. When finding the measurement points, the algorithm does not take into consideration which one of the points belong to the same edge. Finding the correct measurement points could be made more reliable in the future if the edge information was also taken into consideration.

REFERENCES

[1] Lowe DG, Object recognition from local scale-invariant features, Proceedings of the seventh IEEE international conference on computer vision, Vol. 2, 1999, pp.1150-1157

[2] Skrypnyk I and Lowe DG, Scene modelling, recognition and tracking with invariant image features, Third IEEE and ACM international symposium on mixed and augmented reality, 2004, pp.110-119

[3] Panin G and Knoll A, Mutual Information-Based 3D Object Tracking, International Journal of Computer Vision, Vol. 78, No. 1, 2008, pp.107-118

[4] Hinterstoisser S, Benhimane S and Navab N, N3M: Natural 3D Markers for Real-Time Object Detection and Pose Estimation, IEEE 11th International Conference on Computer Vision, 2007, pp.1-7

[5] Fischler M and Bolles R, Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography, Communications of the ACM, Vol. 24, No. 6, 1981, pp.381-395

[6] Lepetit V and Fua P, Monocular model-based 3d tracking of rigid objects: A survey, Foundations and Trends in Computer Graphics and Vision, Vol. 1, 2005, pp.1-89

[7] Canny J, A computational approach to edge detection. IEEE Transactions on pattern analysis and machine intelligence, No. 6, 1986, pp.679-698

[8] Harris C and Stennett C, RAPID-a video rate object tracker, BMVC, 1990, pp.1-6.

[9] Zisserman MAA, Robust object tracking, Proceedings of the Asian Conference on Computer Vision, Singapore, 1995, pp.5-8.

[10] Drummond T and Cipolla R, Real-time visual tracking of complex structures, IEEE Transactions on pattern analysis and machine intelligence, Vol. 24, No. 7, 2002, pp.932-946

[11] Seo B, Park H, Park J, et al. Optimal local searching for fast and robust textureless 3D object tracking in highly cluttered backgrounds, IEEE transactions on visualization and computer graphics, Vol. 20, No. 1, 2013, pp.99-110

[12] Jurie F and Dhome M, Real Time Robust Template Matching, BMVC, 2002, pp.123-132.

[13] Masson L, Dhome M and Jurie F, Robust real time tracking of 3d objects, Proceedings of the 17th International Conference on Pattern Recognition, Vol. 4, 2004, pp.252-255

[14] Jurie F and Dhome M, Real time 3D template matching, Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Vol. 1, 2001

[15] Ladikos A, Benhimane S and Navab N, A real-time tracking system combining template-based and feature-based approaches, VISAPP, 2007, pp.325-332

[16] Gordon NJ, Salmond DJ and Smith AF, Novel approach to nonlinear/non-Gaussian Bayesian state estimation, IEE proceedings F (radar and signal processing), Vol. 140, No. 2, 1993, pp.107-113

[17] Choi C and Christensen HI, 3D textureless object detection and tracking: An edge-based approach, IEEE/RSJ International Conference on Intelligent Robots and Systems, 2012, pp.3877-3884

[18] Liu M, Tuzel O, Veeraraghavan A, et al. Fast directional chamfer matching, IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2010, pp.1696-1703

[19] Christen O, Naroska E, Micheel A, et al. Target marker: A visual marker for long distances and detection in realtime on mobile devices, 2nd International Conference of Machine Vision and Machine Learning, 2015

[20] Hoff WA, Nguyen K and Lyon T, Computer-vision-based registration techniques for augmented reality, Intelligent Robots and Computer Vision XV: Algorithms, Techniques, Active Vision, and Materials Handling, Vol. 2904, 1996, pp.538-548, International Society for Optics and Photonics

[21] State A, Hirota G, Chen DT, et al. Superior augmented reality registration by integrating landmark tracking and magnetic tracking, Proceedings of the 23rd annual conference on Computer graphics and interactive techniques, 1996, pp.429-438

[22] Neumann U, Jongweon L and Youngkwan C, A Multi-ring Color Fiducial System and An Intensity-invariant Detection Method for Scalable Fiducial-Tracking Augmented Reality, Proc. Int'l Workshop Augmented Reality, 1999, pp.147-165

[23] Koller D, Klinker G, Rose E, et al. Real-time vision-based camera tracking for augmented reality applications, Proceedings of the ACM symposium on Virtual reality software and technology, 1997, pp.87-94

[24] Rekimoto J, Matrix: A realtime object identification and registration method for augmented reality, Proceedings. 3rd Asia Pacific Computer Human Interaction (Cat. No. 98EX110), 1998, pp.63-68, IEEE

[25] Nunes JF, Moreira PM and Tavares JMR, Human motion analysis and simulation tools: a survey, Handbook of Research on Computational Simulation and Modeling in Engineering, 2016, pp.359-388, IGI Global

[26] Park Y, Lepetit V and Woo W, Texture-less object tracking with online training using an RGB-D camera, 2011 10th IEEE International Symposium on Mixed and Augmented Reality, 2011, pp.121-126, IEEE

[27] Choi C and Christensen HI, RGB-D object tracking: A particle filter approach on GPU, 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2013, pp.1084-1091, IEEE.

[28] Heikkila J, Geometric camera calibration using circular control points, IEEE Transactions on pattern analysis and machine intelligence, Vol. 22, No. 10, 2000, pp.1066-1077

[29] Slabaugh GG, Computing Euler angles from a rotation matrix, Retrieved on August, Vol. 6, No. 2000, 1999, pp.39-63

[30] Hartley R and Zisserman A, Multiple view geometry in computer vision, Cambridge University Press, 2003.

[31] Haralick RM, Lee C, Ottenburg K, et al. Analysis and solutions of the three point perspective pose estimation problem, CVPR, Vol. 91, 1991, pp.592-598.

[32] Quan L and Lan Z, Linear n-point camera pose determination, IEEE Transactions on pattern analysis and machine intelligence, Vol. 21, No. 8, 1999, pp.774-780

[33] Dementhon DF and Davis LS, Model-based object pose in 25 lines of code, International journal of computer vision, Vol. 15, 1995, Springer

[34] Bradski G and Kaehler A, Learning OpenCV: Computer vision with the OpenCV library, O'Reilly Media, Inc., 2008

[35] Triggs B, McLauchlan PF, Hartley RI, et al. Bundle adjustment—a modern synthesis, International workshop on vision algorithms, 1999, pp.298-372, Springer.

[36] Karlsson R, Schon T and Gustafsson F, Complexity analysis of the marginalized particle filter, IEEE Transactions on Signal Processing, Vol. 53, No. 11, 2005, pp.4408-4411, IEEE

[37] Izadi S, Kim D, Hilliges O, et al. KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera, Proceedings of the 24th annual ACM symposium on User interface software and technology, 2011, pp.559-568

[38] Pérez P, Hue C, Vermaak J, et al. Color-based probabilistic tracking, European Conference on Computer Vision, 2002, pp.661-675, Springer

[39] Wang G, Wang B, Zhong F, et al. Global optimal searching for textureless 3D object tracking, The Visual Computer, Vol. 31, No. 6-8, 2015, pp.979-988, Springer

[40] Comaniciu D, Ramesh V and Meer P, Real-time tracking of non-rigid objects using mean shift, Proceedings IEEE Conference on Computer Vision and Pattern Recognition. CVPR 2000 (Cat. No. PR00662), Vol. 2, 2000, pp.142-149, IEEE