Abstract—In this work, a novel object tracking system is proposed which tracks visual objects on Apple iPhone 4 platform in real-time. The system utilizes the colorspace of the frames provided by iPhone camera, in parallel with the motion data provided by iPhone motion sensors, to cancel the effect of iPhone movements during matching different candidate tracks. The proposed system also adapts to changes in target appearance and size, thus leading to a robust tracking. Several experiments conducted on actual video sequences are used to illustrate the functionality of the proposed approach.

I. INTRODUCTION

Object tracking is the process of determining the trajectory of real-world objects using the frames of a video sequence. Some of its most common applications include automated surveillance of scenes, automated vehicle navigation, perceptual user interfaces for human-computer interaction, augmented reality.

Using a point representation for objects, the task of tracking can be formulated as finding the corresponding points from one frame to the next [1]–[8]. Point Tracking methods are mostly suitable for tracking very small objects which can be represented by a single point. Therefore, point tracking methods are not suitable for our goal which is real-time object tracking on iPhone platform.

In Kernel Tracking methods, [10]–[21] the tracking target is mainly represented as a simple object region, which has parametric motion (for example translation or affine motion) between frames. The tracking problem is then solved by computing the motion of the object, from one frame to the next. Kernel Tracking methods mainly use simple geometric shapes for object representation. Depending on the model considered for object, kernel tracking methods are also usually successful in tracking objects with partial occlusions. Therefore, despite the possible disadvantages of the kernel tracking methods, they are suitable for object tracking on a mobile platform. In this work, we have used the kernel-based object tracking approach proposed by Dorin et al. in [22] as we will discuss in Section II.

II. VISUAL OBJECT TRACKING ON IPHONE

A. Object Tracking Algorithm

We have to obtain a representation for the object that makes it as different as possible from other objects appearing in the frames of video sequence.

1) Target Model: Suppose the tracking target is represented in one of the frames as a \((w \times h)\) rectangle containing \(N\) pixels, \(\{x_i\}_{i=1, 2, \ldots, N}\) as shown in Fig. 1. In order to eliminate the influence of different target dimensions, all pixel locations \(\{x_i\}_{i=1, 2, \ldots, N}\) are shifted so that the target center will have the coordinates \(\vec{O} = (0, 0)\). Moreover, the horizontal and vertical dimensions are independently rescaled with \(w\) and \(h\), respectively, so that a unit circle will reside within the target rectangle shown in Fig. 1.

It is possible to extract several features using the RGB color channels, including image intensity, RGB color values, edges and texture [18], [26], [27]. It this work, the values of the RGB color channels are used as the features for each pixel due to the limitations of the iPhone camera and CPU power. After extracting \(d\) tracking features for each of the \(N\) pixels in the target rectangle, they are mapped into a set \(\{u_i\}_{i=1, 2, \ldots, N}\) of \(N\) feature vectors in \(d\)-dimensional tracking feature space.

In order to satisfy the low-computational cost imposed by real-time processing, this multidimensional feature space is first quantized using a mapping function,

\[B: \mathbb{R}^d \rightarrow \{1, \ldots, m\}, \tag{1}\]

into \(m\) bins represented by \(u = 1 \ldots m\). Thus, The function \(B(u)\) associates to a feature vector \(\{u_i\}_{i=1, 2, \ldots, N}\) the index \(B(u)\) of its bin in the quantized feature space, and \(u\) represents...
the index of the bins in the d-dimensional feature space. Each color channel in iPhone frames is represented by 8 bits. So each color channel can take one of the 256 values in the range \( \{0, \ldots, 2^8 - 1\} \). Every channel has been divided into 16 bins. Therefore, the total number of bins in the multidimensional feature space is

\[
m = 16 \times 16 \times 16 = 4096
\]

In order to take into account the spatial arrangement of pixels in the target region, an isotropic spatial kernel \( K(x) \), with a convex and monotonic decreasing kernel profile

\[
k : [0, \infty) \rightarrow R,
\]

such that \( K(x) = k(\|x\|^2) \), is used to weight all pixels \( \{x_i\}_{i=1,2,\ldots,N} \) in the target region with weights equal to \( k(\|x_i\|^2) \). There are several possible choices available for this spatial kernel [12]. In this work, we have used the Epanechnikov kernel profile, \( K_E(\cdot) \), which has been recommended by the authors in [22].

Finally, the target model,

\[
\hat{q} = \{\hat{q}_u\}_{u=1}^m, \quad \sum_{u=1}^m \hat{q}_u = 1 \quad (2)
\]

is built as the histogram-based estimation of probability density function for the tracking feature space. Thus, the value of \( \hat{q}_u \) is the probability of the features in bin \( u = 1 \ldots m \) at the target region, which is computed using Eqn. 3.

\[
\hat{q}_u = C_t \sum_{i=1}^N k(\|x_i\|^2) \delta [B(u_i) - u] \quad (3)
\]

In Eqn. 3, \( \delta [\cdot] \) is the Kronecker delta function, and \( C_t \) is the normalization constant.

2) Target Localization: Suppose that we are given the next frame in which we wish to localize the target (i.e. update the target state). Consider one candidate region centred at \( y \) containing \( M \) pixels, \( \{y_i\}_{i=1,2,\ldots,M} \). Again, in order to eliminate the influence of different target dimensions, the horizontal and vertical dimensions for pixel locations \( \{y_i\}_{i=1,2,\ldots,M} \) are independently rescaled with respect to the target width and height \((w,h)\), respectively. \( M \) feature vectors \( \{v_i\}_{i=1,2,\ldots,M} \) are extracted in the \( d \)-dimensional feature space in the same way it was done for the pixels within the target region in Section II-A1. A model for this candidate is extracted in exactly the same way it was done for the target in Section II-A1. The candidate model is shown in Eqn. 4.

\[
\hat{p}_u(y) = C_c \sum_{i=1}^M k\left(\left\|\frac{y - y_i}{h_c}\right\|^2\right) \delta [B(v_i) - u] \quad (4)
\]

All parameters in Eqn. 4 are exactly the same as in Eqn. 3. The difference here is that a larger number of pixels are used in the evaluation of the candidate model than the target model (i.e. \( M \geq N \)). Therefore, the bandwidth \( h_c \) is used in the candidate model in order to include all pixels in the larger area. \( C_c \) in Eqn. 4 is the histogram normalization constant.

There are many similarity measures available [13]. In this work, the sample based Bhattacharyya coefficient suggested by Dorin et al. in [22] has been used as the similarity measure between the target and candidate models. Sample based Bhattacharyya coefficient between two sample based distribution \( p(y) \) and \( q \) is defined as shown in Eqn. 5.

\[
\hat{\rho}(y) = \hat{\rho} [p(y), q] = \sum_{u=1}^m \sqrt{\hat{p}_u(y)\hat{q}_u} \quad (5)
\]

The estimate to the gradient of similarity measure in Eqn. 5 is achieved as shown in Eqn. 6.

\[
\begin{align*}
\nabla \hat{\rho}_2(y) &= \left[ \sum_{i=1}^M w_i g \left( \left\| \frac{y - y_i}{h_c} \right\|^2 \right) \right] \\
& \times \left[ \sum_{i=1}^M w_i y_i g \left( \left\| \frac{y - y_i}{h_c} \right\|^2 \right) - y \right] \\
& \sum_{i=1}^M w_i g \left( \left\| \frac{y - y_i}{h_c} \right\|^2 \right)
\end{align*}
\]

Where \( g(x) = -K'(x) \).

Therefore, starting from a point \( y^{old} \), the new estimate of the local maximum of \( \rho_2(y) \) can be obtained by iterating over Eqn. 7.

\[
y^{new} = \frac{\sum_{i=1}^M Y_i w_i g \left( \left\| \frac{y^{old} - y_i}{h_c} \right\|^2 \right)}{\sum_{i=1}^M w_i g \left( \left\| \frac{y^{old} - y_i}{h_c} \right\|^2 \right)}
\]

This procedure is guaranteed to converge to a stationary point provided that the kernel \( K \) has a convex and monotonically decreasing profile [12]. Therefore, the exhaustive search for the mode of the similarity measure \( \hat{\rho}(y) \) in (5) can be replaced with mean-shift iterations.

Fig. 2 shows the results of applying the visual tracking algorithm to the frames of an iPhone sequence for tracking the robot. Fig. 3 shows the number of iterations used for target localization in each frame in Fig. 2. For frames where larger movements of objects has occurred, more iterations has been used. However, as shown, the number of iterations has never exceeded 5. Fig. 4 shows the values of the Bhattacharyya Coefficient (5) obtained after localizing the target in each frame in Fig. 2. The value of this similarity measure has been significantly decreased around frame number 75, which is due to the changes in illumination conditions for the target. The values for Bhattacharyya Coefficient are small between frames 100-150 as well. This has happened because of the rotations of the target, which make the appearance of the candidates less similar to the target appearance.

A groundtruth for the target location has been evaluated by manually selecting the target position in each frame. Fig. 5 shows the resulting errors (in pixels unit) for tracking the robot in Fig. 2. It could be seen that the tracker has not lost the target at all. However, there are some small tracking drifts after frame number 100 which is caused because of the fast movements and rotations of the object in the sequence.
Fig. 2. Results of applying visual tracking algorithm to an iPhone sequence for tracking the robot. Frames 0, 50, 100, 150 of 222 frames are shown.

Fig. 3. Number of iterations used for target localization in each frame in Fig. 2.

Fig. 4. Values of the Bhattacharya Coefficient (5) obtained after localizing the target in each frame in Fig. 2.

Fig. 5. Resulting tracking errors (in pixels unit) for tracking the robot in Fig. 2.

Fig. 6. Screen shots of the iPhone object tracking application based on visual tracking algorithm. Bhattacharya Coefficient and frame rate are mentioned in each screen shot.

B. Adapting to Target Changes

Target appearance may change in so many ways during tracking. Some of these changes are temporary, such as partial occlusions, shape deformations and sudden illumination changes. On the other hand, there are changes that are not temporary, such as permanent changes in illumination conditions and changes in target size. In order to keep up with permanent target changes, the tracking system must detect such evolutions and adapt to them.

1) Updating Target Size: The size of the tracking target in the frames often changes over time. Denote by \((w_{\text{prev}}, h_{\text{prev}})\) the size used for target model (3) in previous frame. Changes in target size are updated by computing a target model (3) at current location for multiple different values of \((w, h)\). The Bhattacharya coefficient (5) between all these models and the target model \(\hat{q}\) is then evaluated. The best target size, \((w, h)^*\), yielding the maximum Bhattacharya coefficient, is retained. The new target size \((w_{\text{new}}, h_{\text{new}})\) is obtained through filtering in order to avoid over sensitive bandwidth changes.

This approach is similar to what Dorin et al. suggested in [22]. They suggested to run the localization process three times for each frame in order to find the optimum bandwidth. Since there are no parallel processing resources available on iPhone, this approach will linearly decrease the resulting iPhone tracking frame rate. In our work, the target size is updated by only evaluating \(\hat{\rho}(y)\) in (5) for different bandwidths at current target location. Thus, no additional localization tasks are performed for each frame and the resulting iPhone tracking frame rate will not decrease.

2) Updating Target Model: After updating the target size in section II-B1, the target model (3) is also updated through filtering.

3) Adaptive Frame Resizing: The models used for target localization in (3) and (4) are based on the histogram of the tracking feature space. Thus, resizing the image will not affect the model as long as enough number pixels are available for the histogram evaluation.
Fig. 7. Results of applying adaptive visual tracking algorithm to an iPhone sequence in order to track the robot. Frames 0, 40, 90 and 110 of 120 frames are shown.

Therefore, in order to avoid variations in the iPhone tracking frame rate, each frame is first resized according to the current size of the target, prior to tracking.

Fig. 7 shows the results of applying the adaptive visual tracking algorithm to an iPhone sequence in order to track the robot. As shown, the tracker is not only updating the position of the target in frames, but also it updates the size of the target as the target becomes closer or further to the iPhone.

Fig. 8 shows the values of the Bhattacharya Coefficient (5) obtained after localizing the target in each frame in Fig. 7. The Bhattacharya Coefficient has reduced between frames 50-80 which is because of the rotations of iPhone and motion blurring in the frames.

Fig. 9 shows screen shots of the iPhone object tracking application based on adaptive visual tracking algorithm where the moving tooth brush is being tracked. As shown, The object has been localized in all screen shots independent of the changes happening in the background objects. Moreover, changes in the target size and appearance have also been tracked by the application. The resulting Bhattacharya Coefficient and frame rate are mentioned in each screen shot. As shown, the resulting frame rates are large enough to be considered as real-time tracking.

In order to investigate the effect of adaptive frame resizing described in section II-B3, the iPhone tracking application has been applied to a squared shape object in two situations: without frame resizing, and with frame resizing. For both cases, the resulting tracking frame rates for different sizes of the target have been computed. Fig. 10 shows the resulting iPhone frame rates. As shown, when the frames are processed in full resolution for all target sizes, then the resulting tracking frame rates on iPhone will significantly decrease as the target becomes larger. But with the adaptive frame resizing approach in section II-B3, the tracking frame rate on iPhone will always be between 15-30 FPS, which leads to real-time and live results. Therefore, with the adaptive frame resizing, a better object tracking frame rate has been achieved, while having the same object tracking accuracy.

III. SPATIAL UPDATING ON iPHONE

Previous neurological studies have demonstrated that humans and monkeys update the location of visual targets based on the head movements [29]–[31]. In other words, the information about the movements of the head is used as a feedback source in the human visual system, even in the absence of visual feedback. Therefore, the brain uses the head motion information along with the visual information in order to keep track of the objects. As a simple example, it is much easier to keep track of the words on a piece of paper when the head moves, rather than when the paper is moving. This is because in the first case the head motion feedback helps the brain in object tracking, while in the second case there is no such feedback about the movement of object in the visual field. This ability is referred to as Spatial Updating.

In this section, first, a complete geometric camera calibration model for mappings between the world and iPhone visual field is described. Then, an approach for utilizing iPhone motion sensors in Spatial Updating is proposed. Finally, the proposed Spatial Updating algorithm is combined with the object tracking method in Section II in order to improve the performance of iPhone object tracking and achieve the best possible results on iPhone in real-time.
A. Camera Calibration Maths

1) Coordinate Frames: Consider a real object in the 3D world. We consider the position of this object in three coordinate systems:

- A world coordinate system, $\vec{X}_w$, which is determined by considering a constant standard 3D orthogonal basis in world.
- A camera coordinate system, $\vec{X}_c$, where the origin is at the nodal point of the camera and the $z$-axis is taken to be the optical axis of the camera (with points in front of the iPhone having a positive $z$ value).
- An image coordinate system, $\vec{p}$, where $p_1$ and $p_2$ are the horizontal and vertical pixel coordinates in the image, respectively.

Therefore, the formation of objects in the video requires a mapping from $\vec{X}_w$ to $\vec{p}$ for every frame. In order to do so, two mappings are required. The first mapping is from world to camera coordinates, which is called external calibration, as will be discussed in section III-A2. The second mapping is from camera to image coordinates, which is called internal calibration, as will be discussed in section III-A3.

2)External Calibration: The external calibration parameters specify the 3D coordinate transformation from world to camera as shown in Eqn. 8.

$$\vec{X}_c = \mathbf{R} (\vec{X}_w - \vec{d}_w). \quad (8)$$

In Eqn. 8, $\vec{d}_w$ is the location of the camera coordinates origin (camera nodal point), in world coordinates, and $\mathbf{R}$ is the $3 \times 3$ rotation matrix from world coordinates to camera coordinates.

The external calibration parameters are commonly written in terms of a $3 \times 4$ matrix $\mathbf{M}_{\text{ext}}$:

$$\vec{X}_c = \mathbf{M}_{\text{ext}} \begin{pmatrix} \vec{X}_w \\ 1 \end{pmatrix}, \quad (9)$$

where

$$\mathbf{M}_{\text{ext}} = \begin{pmatrix} \mathbf{R} & -\mathbf{R} \vec{d}_w \end{pmatrix}. \quad (10)$$

Thus, the position of the object in the camera coordinate frame is found by changing the basis of coordinates from world coordinate system to camera coordinate system.

3) Internal Calibration: The position of a 3D point in camera coordinate system,

$$\vec{X}_c = \begin{pmatrix} X_{1c} \\ X_{2c} \\ X_{3c} \end{pmatrix},$$

on the image plane is given by perspective projection [32]:

$$\vec{x}_c = \frac{f}{X_{3c}} \begin{pmatrix} X_{1c} \\ X_{2c} \end{pmatrix} = \begin{pmatrix} x_{1c} \\ x_{2c} \end{pmatrix}, \quad (11)$$

where $f$ is called the focal length of camera, i.e. the orthogonal distance between the nodal point of the camera and the image plane.

Transforming the 3D image position $\vec{x}_c$ to pixel coordinates $\vec{p}$ is done using the internal calibration as shown in Eqn. 12.

$$\vec{p} = \begin{pmatrix} \frac{1}{p_w} & 0 & \frac{a_1}{p_w} \\ 0 & 1 & \frac{a_2}{p_h} \\ 0 & 0 & \frac{f}{1} \end{pmatrix} \vec{x}_c = \mathbf{M}_{\text{int}} \vec{x}_c \quad (12)$$

In Eqn. 12, it is assumed that the pixels on the iPhone image plane are of size $p_w \times p_h$, and $(a_1, a_2)$ is the intersection of the optical axis with the image plane, in pixel coordinates. The $3 \times 3$ matrix $\mathbf{M}_{\text{int}}$ is called the internal calibration matrix.

The inverse of internal calibration is also possible which is to convert a point $\vec{p}$ in image coordinate system, to the camera coordinate system, $\vec{x}_c$, as show in Eqn. 13.

$$\vec{x}_c = \mathbf{M}_{\text{int}}^{-1} \vec{p} = \begin{pmatrix} p_w & 0 & -a_1p_w \\ 0 & p_h & -a_2p_h \\ 0 & 0 & f \end{pmatrix} \vec{p} \quad (13)$$

The next step is to transform $\vec{x}_c$ to the image coordinate system, $\vec{X}_c$, using the inverse of the perspective projection given in Eqn. 11, as shown in Eqn. 14.

$$\vec{X}_c = \frac{f}{X_{3c}} \vec{x}_c = \begin{pmatrix} X_{1c} \\ X_{2c} \\ X_{3c} \end{pmatrix} \quad (14)$$

Therefore, by knowing the internal calibration parameters, it is possible to convert the position of an object between the camera coordinate system and image coordinate system.

In this work, frames are captured by iPhone at a size of $640 \times 480$ pixels. Therefore, the values for $a_1$ and $a_2$ in the internal calibration parameters are 320 and 240, respectively. The values for $p_w$ and $p_h$ are both approximately 7 mm. The focal length of iPhone 4 camera is also $f = 3.85$ mm.

Therefore, the values for $\mathbf{M}_{\text{int}}$ and $\mathbf{M}_{\text{int}}^{-1}$ in Equations 12-13 do not change through all of our experiments on PC and iPhone and are shown in Equations 16-17 below.

$$\mathbf{M}_{\text{int}} = 10^5 \times \begin{pmatrix} 1.4286 & 0 & 0.8325 \\ 0 & 1.4286 & 0.6247 \\ 0 & 0 & 0.0026 \end{pmatrix} \quad (16)$$

$$\mathbf{M}_{\text{int}}^{-1} = 10^{-4} \times \begin{pmatrix} 0.07 & 0 & -22.43 \\ 0 & 0.07 & -16.83 \\ 0 & 0 & 38.50 \end{pmatrix} \quad (17)$$

B. Spatial Updating

In this section, an algorithm is proposed for performing spatial updating on an iPhone. Then, an approximate version of this algorithm is suggested for implementation on Apple iPhone 4 using its motion sensors.

1) Spatial Updating Procedure: Here we consider two consecutive frames, namely the previous and current frames. Then, we evaluate the changes that happen in an object’s position in the image coordinate system due to iPhone’s movements (the same as the Spatial Updating in the human brain). This is done by means of updating the world coordinate system as well as the whole camera calibration procedure (section III-A), between two consecutive frames, as shown in Eqn. 18.
The error of the spatial updating estimation depends on the displacements made by iPhone and the object in 3D world, relative to their position at the point where an accurate value for \( \bar{p} \) was available. Therefore, in this section, we use the the Adaptive Mean-Shift Tracking method proposed in Section II, in conjunction with the approximate Spatial Updating. This way, the estimate to object position \( \hat{p} \) obtained form the approximate Spatial Updating procedure is used for initialization of the Adaptive Mean-Shift tracking algorithm in order to speed up the whole tracking process.

The Spatial Object Tracking method has been applied to an iPhone sequence in order to track the robot. Frames 0, 50, 100, 150 and of 222 frames are shown.

\[
\hat{p}_{\text{cur}} = \left( \begin{array}{cc}
X_{\text{cur}}^{\text{pre}} \\
X_{\text{cur}}^{\text{int}}
\end{array} \right) M_{\text{int}} R M_{\text{int}}^{-1} p_{\text{pre}} - \left( \frac{f}{X_{\text{cur}}^{\text{int}}} \right) M_{\text{int}} R \hat{\Delta}_d \tag{8}
\]

Therefore, given the image position of a stationary object in the previous frame, \( p_{\text{prev}} \), it is possible to estimate the image position of the object for the current frame if we know the object depth for previous frame, \( X_{\text{cur}}^{\text{pre}} \), plus the rotation matrix, \( R \), and the displacement, \( \hat{\Delta}_d \), of iPhone between these two frames. For very small movements of the device between frames, \( X_{3,c} \) has negligible effect. Thus, a constant value of 1.0m has been used for \( X_{3,c} \) in this work.

A simplified version of the Spatial Updating has been implemented on the Apple iPhone. A demo of the proposed Real-Time Spatial Updating as an iPhone application is available at http://youtu.be/GHeOffqc4KE.

### C. Combining Spatial Updating with Object Tracking

The error of the spatial updating estimation depends on the displacements made by iPhone and the object in 3D world, relative to their position at the point where an accurate value for \( \bar{p} \) was available. Therefore, in this section, we use the the Adaptive Mean-Shift Tracking method proposed in Section II, in conjunction with the approximate Spatial Updating. This way, the estimate to object position \( \hat{p} \) obtained form the approximate Spatial Updating procedure is used for initialization of the Adaptive Mean-Shift tracking algorithm in order to speed up the whole tracking process.

The Spatial Object Tracking method has been applied to an iPhone sequence to track the robot. Fig. 11 shows the results of this object tracking. The estimate to the object position in current frame, provided by the Approximate Spatial Updating procedure has been useful in object tracking for 155 out of 222 frames in this video sequence. Thus, Spatial Updating has been useful 70% of the time.

Fig. 12 compares the resulting tracking errors for tracking the robot in the first 100 frames of an iPhone sequence.

The dotted curve is obtained when only the spatial updating procedure (Section III-B1) is used. The average error in this case is 25.2 pixels. Obviously, the tracking error is very large for this case since the visual info are not used at all and the tracking is only based on the movements of the device. The dashed curve is obtained for the Mean-Shift Object Tracking (i.e. when Spatial Updating is not used in object tracking) with an average tracking error of 8.2 pixels. Finally, the solid curve in Fig. 12 is obtained for the combined Spatial Object Tracking. An average tracking error of 5.0 is achieved for the combined Spatial Object Tracking. This means that utilizing the motion information provided by the iPhone has improved the performance of tracking in terms of tracking accuracy as well.

Figures 13 - 14 show the results of the same experiments for comparing the three approaches, as described before. The graphs in Fig. 13 is obtained when tracking the violin in an iPhone sequence. The graphs in Fig. 14 is obtained when tracking the robot in an iPhone sequence.

The numerical results for all three experiments are listed in Tables I - II. As shown in Table I, the utilization of the motion information in object tracking has decreased the average number of required mean-shift iterations, thus leading to faster tracking results (higher tracking frame rate). As shown in Table II, combining spatial updating with visual tracking has decreased the average tracking error. In other words, it can lead to more accurate object tracking results. Therefore, the combined spatial object tracking method has achieved the best object tracking results in terms of both speed and accuracy.
Fig. 14. Comparing the resulting tracking errors.

Fig. 15 shows screen shots of the iPhone object tracking application based spatial object tracking where similar to Fig. 6 the blue box is being tracked. However, the difference is this experiment is that the iPhone has been rotating much faster than in Fig. 6. As shown, the Spatial Updating procedure makes it possible to keep track of the object in frames even when the iPhone rotating fast. Moreover, Utilizing the motion information provided by the iPhone has improved the performance of tracking because the iPhone is now aware of its own rotations and takes them into account. The resulting Bhattacharya Coefficient and frame rate are mentioned in each screen shot. Again, the tracking frame rates are large enough to be considered as real-time tracking. A demo of the proposed Real-Time Spatial Object Tracking as an iPhone application is available at http://youtu.be/YlHpZ_1qx0A.

Table I. Comparison of the average number of used mean-shift iterations.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Visual Tracking</th>
<th>Spatial Object Tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 12</td>
<td>1.8</td>
<td>0.5</td>
</tr>
<tr>
<td>Fig. 13</td>
<td>1.4</td>
<td>0.3</td>
</tr>
<tr>
<td>Fig. 14</td>
<td>4.2</td>
<td>2.2</td>
</tr>
</tbody>
</table>

Table II. Comparison of the average tracking error (pixels).

<table>
<thead>
<tr>
<th>Experiment</th>
<th>Visual Tracking</th>
<th>Spatial Updating</th>
<th>Spatial Object Tracking</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fig. 12</td>
<td>8.2</td>
<td>25.2</td>
<td>5.0</td>
</tr>
<tr>
<td>Fig. 13</td>
<td>7.6</td>
<td>28.0</td>
<td>4.5</td>
</tr>
<tr>
<td>Fig. 14</td>
<td>49.6</td>
<td>87.1</td>
<td>20.8</td>
</tr>
</tbody>
</table>

IV. Conclusion

In this work, we proposed and implemented an approach for performing accurate object tracking on iPhone platform in real-time. Regarding the characteristics of the iPhone CPU and camera, we chose the Kernel-Based Object Tracking method [22] as the base object tracking algorithm in our work. Then, we added some steps to the Kernel-Based Object Tracking Algorithm in order to make it robust to the changes in the appearance of objects over time. In order to further utilize the sensors available on iPhone, we also proposed a novel Spatial Updating approach for sensing iPhone’s motion when capturing frames, and cancelling the effect of camera motion in object tracking. Finally, we combined the proposed Adaptive Object Tracking Algorithm with the Spatial Updating procedure to take advantage of both visual and motion sensors of iPhone for performing an accurate object tracking in real-time. Experimental results in this work show using the proposed Spatial Updating approach has the potential to significantly improve the performance of object tracking on iPhone if used in conjunction with object tracking.

References

[33] D. Goehl, D. Sachs, Motion sensors gaining inertia with popular consumer electronics, InvenSense, Inc.