Abstract

Throughout the years, Robotics has shown great benefits to humankind, as they helped us explore other planets and safely defuse explosives. They can also provide assistive care to humans. A robotic manipulator could greatly help the elderly and physically impaired people in providing a more independent lifestyle. But such manipulator must be intuitive and easy to use. This work proposes and tests a system that simplifies this human-robot interaction. This system uses the pose of its user’s head to understand which object he wants to use. By facing the object of interest, the user tells the system that he wants to use that object and the system must prepare the robotic arm to perform the required action for that specific object.

Keywords: head pose estimation, face tracking, point cloud, table detection

Introduction

There are people with autonomous difficulties that would benefit from this technology like the elderly and people with motor difficulties. For these people, a robotic manipulator could assists them during meals or help them get their phones when needed.

However, such robot requires an intuitive input system so that the user can easily and accurately use it. If not, manipulating this arm would be a harder task than its objective. Therefore, the idea of using the user’s sight came to mind with this project. Just as people may point with their fingers to a specific point of interest, the user uses their sight to point to an object of interest.

This system was designed to work in an environment where the user is sitting at a table that contains the objects that he may want to use, like a cellphone or a mug. To perceive this environment, the system uses a Kinect camera. This device is composed of 2 cameras. One can take colour (RGB) images and the other one can take depth images. Together, these images can create a 3D reconstruction of the captured environment.

includes the point cloud of the face, which must be classified in order to compute the users head pose through out the usage of the robot. Subsequently, with the pose, the line of sight can be estimated. In order to know where the line of sight lands on the table, the system must find the point where these two intersect. Additionally, the system also needs to correctly classify the point cloud of the table in order to compute the intersection point. Finally, the distance between the intersection and an object is checked. If its values lies below a certain threshold, the system will assume that the user intends to use that object and must prepare the robotic arm to performed the required commands to correctly use a specific object.

This paper began with an introduction of this system. The next section presents other input systems that are suggested by other people and tells their strengths and disadvantages. Section 3 elaborates more about how the system works by explaining the strategies and methods implemented. In section 4 it is displayed the experimental results of the proposed implementation, followed by the conclusions of the results in section 5.

Background

Dev Anand et al.[5] proposed a system composed of an electric powered wheelchair, a robotic arm called MANUS and a computer. This system is
called FRIEND and uses speech recognition to receive commands from its user. To avoid misinterpretations from noise or misspelled words, the user must provide a certain sequence of words so that the manipulator can perform the desired action. The results of this project showed that this type of input could cause a considerable time delay between the order of the user and the action of the manipulator. Moreover, since some orders may require more keywords, it is more probable that at least one of the words may be misinterpreted and, therefore, the action may not be performed.

Hochberg et al. [7] tested how well a robotic arm can be neurally controlled by its user. This system was tested by two tetraplegic people, who were able to perform actions that were unable to do alone, like drinking a beverage. However this solution utilises a sensor that was implanted on both users primary motor cortex. Therefore is not a very desirable solution for the majority of people, being more of a solution destined specifically to tetraplegic people.

Bassily et al. [4] use a Leap Motion Controller. According to [4, 12], this controller is composed of 3 IR lights and 2 IR cameras, which can detect and track hand and finger motions. With the help of other sensors, this controller lets the user remotely control a robotic arm by making it mimic the gesture that the user is making.

To distinguish itself from these solutions, this paper’s work presents a more intuitive and simple manipulation. It only requires the user to face the object of interest, as if he was pointing at it and saying that he wants to use that object. Besides, this system only needs a Kinect camera and a computer that processes information from it as equipment. Since it is not very uncommon for a person to have a computer, the camera is the only extra accessory needed to be present on the table or near it.

Proposed Methods

3D Data

The system receives, from a Kinect camera, a colour image (RGB) and a depth image in order to extract 3D information. Both images have 640x480 pixels. We chose this camera as it is a commercially available depth camera. This camera must be positioned on a table, around 1 meter away from the user and around the same height as the users head, as figure 1 shows. It also needs to stay in the same place throughout its use. This positioning enables the camera to capture the face and the table on the same image.

A 3D reconstruction of the environment is made of a point cloud generated with, at least, a depth image, as RGB images only provide colour information. This 3D representation is written with the depth camera’s frame as the reference frame.

The depth image stores values that represent the depth \(d\) of each pixel. When converting these pixels into 3D points, these depth values give information about each points’ \(z\) coordinate. Equation (1) shows how this information is generated, where \(\alpha\) and \(\beta\) are intrinsic parameters of the depth camera. Note that \(z\) increases with \(d\) since \(\alpha\) is negative and \(\beta\) is positive.

\[
z = \frac{1}{d.\alpha + \beta} \tag{1}
\]

Afterwards, \(x\) and \(y\) coordinates are extracted by taking in consideration the pixel coordinate \((u,v)\) of each depth value. To perform the rest of the transformation, it is required to use the intrinsic parameters of the infra-red camera.

\[
\begin{align*}
x &= \frac{u - c_u}{f_{sv}} \ast z \\
y &= \frac{v - c_v}{f_{sv}} \ast z 
\end{align*} \tag{2}
\]

where \(f_{sv}, c_u\) and \(c_v\) are the intrinsic parameters of the depth camera. Since we pretend to use the depth camera’s frame as the world frame, no further operation is required.

However, the face detection problem can only be solved by providing a RGB image to the face detector. Since both images are not taken from the same position, a pixel pair will not share the same coordinates in both images. Therefore we need to find the correct colour-depth pixel pairing for these points. The strategy is to convert the environment’s point cloud into pixels but considering that the depth image was taken from the RGB camera’s position. So the first step is to rewrite this point cloud from the RGB camera’s frame perspective. That is performed by using a rotation matrix and a translation vector \((R_{RGB} \, \text{and} \, T_{RGB})\).

\[
E_{RGB} = R_{RGB}E_{depth} + T_{RGB}J_{1,N} \tag{3}
\]

Where \(E_{depth}\) and \(E_{RGB}\) represent the point cloud written from the depth camera’s frame and RGB camera’s frame respectively. The matrix \(J\) represents a matrix whose elements are all equal to 1. In this case, it represents a row of ones repeated \(N\) times, where \(N\) represents, in this case, the total number of points in the point cloud. Next, each point of \(E\) is converted into a pixel coordinate.
system by using each point’s 3D coordinates. Equation (4) shows this conversion for a 3D point with \((x, y, z)\) as coordinates values.

\[
\begin{align*}
\underline{u_1} &= \frac{x}{z} \\
\underline{v_1} &= \frac{y}{z}
\end{align*}
\]

\(u_1\) and \(v_1\) are only auxiliary variables that help to find the real values of \(u\) and \(v\) of this example point. It is still needed to use the intrinsic matrix of the RGB camera \(K_{RGB}\) to get the real values of \(u\) and \(v\). But, for this camera, it is also required to take into account its geometrical model before using \(K_{RGB}\). [9].

\[
\begin{bmatrix}
\underline{u_2} \\
\underline{v_2}
\end{bmatrix} = (1 + k_1r^2 + k_2^2 + k_3^6) \cdot \begin{bmatrix}
\underline{u_1} \\
\underline{v_1}
\end{bmatrix} + \begin{bmatrix}
2k_3u_1v_1 + k_4(r^2 + 2 + u_1^2) \\
k_3(r^2 + 2 + v_1^2) + 2k_4u_1v_1
\end{bmatrix}
\]

\(r^2 = u_1^2 + v_1^2\)

(5)

The first term of the sum represents the radial distortion while the second term is the tangential distortion. \(u_2\) and \(v_2\) also represent auxiliary variables. By applying the intrinsic matrix of the RGB camera, the pixel location in the RGB image of the example point is finally found.

\[
\begin{bmatrix}
\underline{u} \\
\underline{v}
\end{bmatrix} = \begin{bmatrix}
gs_u & 0 & b_u \\
a & gs_v & b_v
\end{bmatrix} \cdot \begin{bmatrix}
\underline{u_2} \\
\underline{v_2}
\end{bmatrix}
\]

(6)

Since we only need to locate the 3D points of the face, it is not required to make this pairing to all points of \(E\). However, to detect potential conversion errors, we applied this conversion to all points during the development of this work to be able to view the point cloud with colour.

Face Detection

The system needs a tool to identify which points represent the face in order to get its pose. Two face detectors were compare. The first detector that was tested was Viola-Jones [11] and the other was MTCNN [13].

Viola-Jones uses a cascade of strong classifiers in order to classify which sub-window of the image contains a face. The cascade’s first classifiers are simpler than the latter ones as it lets the majority of sub-windows, that do not contain a face, to be discarded. Each classifier is trained by a implementation of AdaBoost training and each sub-window is treated as an Haar-like feature, whose values are quickly computed thanks to the integral image method, which is also proposed by [11] work. We tested a Matlab implementation [1] of this detector that can also detect specific face features such as eyes and mouths.

MTCNN is composed of 3 Neural Networks. After creating a image pyramid of the current frame, MTCNN uses its first network, the Proposal Network (P-Net), where it generates candidate sub-windows that may contain a face. Afterwards, the Refine Network (R-Net) rejects the majority of the the false candidates. Lastly, the O-net discards more false candidates and places 5 landmarks to the face (1 for each eye, 1 for the nose and 1 for each corner of the mouth).

Neither detectors could consistently show better face detection results than the other. However, MTCNN was chosen instead of Viola-Jones as it consistently showed more accurate results regarding face features detection, which is important for the face tracking problem.

Face Tracking

The strategy used to solve the face tracking problem was to compare each frame’s face point cloud with another face point cloud with an already known pose. By finding the common keypoints between these two point clouds, it is possible to discover the required transformation that make these point clouds fit each other. The current pose of the face can be found by applying this transformation to the already known pose (reference pose). Figure 3 shows an overview this strategy.

![Figure 3: Overview of the implemented solution for the face tracking problem](image)

The first step is to choose a depth image and a RGB image to create the face model. These images can be the first images recorded as long as they show the user facing the camera. Then, the face must be detected on the reference RGB image.
After detection, it is needed to identify the 3D points of the environment’s point cloud that represent the 3D reconstruction of the face model. Only a sample of these points is required to create the reference pose, with a minimum of 3 keypoints. It is possible to use the keypoints detected by MTCNN, but it would be advantageous to have a bigger sample to avoid any disturbance caused by a keypoint with a badly captured depth. So an implementation of SIFT was also used [10, 8]. SIFT detects keypoints in a RGB image regardless of the scale and angle of the image. This means that, it is able to detect the same keypoint of the face in other images. SIFT detects around 20 30 face keypoints per frame. Afterwards, it is found the respective 3D points of these keypoints.

To create the reference pose it is assumed that the face is approximately a plane and that its normal vector ($s_m$) can define the reference orientation while the point cloud’s centre ($p_m$) can define the position of the face model. This approximation is the reason why the face must be facing the camera when building the face model. The normal vector could come out of the cheek instead of the nose area, if the user were to look away from the camera. The keypoints are sampled and analysed by a implementation of RANSAC [2]. Generally, RANSAC samples a specific number of points and tries to find the parameters of a specified model that contain those points. Then it checks, from the remaining points, how many can fit into this model (inliers). RANSAC needs to repeat these steps several times, with other samples, to increase the probability of getting a good approximation of the pretended parameters of the model. In the end, RANSAC outputs the parameters that made the model contain the most inliers. In this case, the model is the plane model, which is displayed in equation (7). The last parameter of the plane is considered to be equal to one for simplification.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix}^T \cdot s_m + 1 = 0$$

(7)

For a sample $C$ of $N$ keypoints, it is pretended to find the normal vector $s_m$ that solves the following equation.

$$\min_{s_m} \|C^T \cdot s_m - J_{N,1}\|^2$$

(8)

By derivating this expression, it is possible to find the solution that minimises the original function.

$$2C^T \cdot (C \cdot s_m - J_{N,1}) = 0 \Leftrightarrow s_m = (C^T \cdot C)^{-1} \cdot C^T \cdot J_{N,1}$$

(9)

After getting its pose, the face model is finally ready to be compared with other frames’ faces. The face is detected in each frame and SIFT detects keypoints on the face. The objective is to find out which of these keypoints is also present in the face model. These matches are also performed by SIFT and they are used to find the transformation that minimises the total distance between the points of each match.

$$\begin{align*}
\min_{R,T} & \|F - (R \cdot M + T \cdot J_{1,N})\|^2 \\
\text{st.} & \quad R^T \cdot R = I \\
& \quad \det(R) = 1
\end{align*}$$

(10)

where $F$ and $M$ represent samples of the keypoints of the currently analysed frame and the face model respectively. Both samples are composed of up to 6 SIFT keypoint pairs and the available pairs of the face detector keypoints, making a maximum of 11 pairs. It is proposed Fischler, Arun et al. [3, 6] to use singular value decomposition (SVD) to get the solution of a least squares problem that contains rotation and translation transformations. First we can find the translation vector by using both samples’ centres ($\overline{f}$ and $\overline{m}$).

$$\overline{f} = \frac{1}{N} \sum_{i=1}^{N} f_i$$

$$\overline{m} = \frac{1}{N} \sum_{i=1}^{N} m_i$$

(11)

(12)

This last expression helps simplify equation (8):

$$\begin{align*}
\min_{R} & \|F_d - R \cdot M_d\|^2 \\
\text{st.} & \quad R^T \cdot R = I \\
& \quad \det(R) = 1
\end{align*}$$

(14)

where $F_d$ and $M_d$ represent the distances between the sample points to their respective sample’s centre. Each distance is written as a vector. Afterwards, the expression is expanded, according to the following expression:

$$F_d^T \cdot F_d - 2F_d^T \cdot R \cdot M_d + M_d^T \cdot M_d$$

(15)

Only the second term of this expression is non constant as the other terms only depend of the sample points. So to minimise this problem it is required to maximise this term, which is the same as maximising its trace [6]. By using SVD on the resulting matrix on the left side of equation (14), it is possible to get the matrices that create the pretended rotation matrix.

$$M_d \cdot F_d^T = U \cdot \Lambda \cdot V \Rightarrow R = V \cdot U^T$$

(16)
With \(R\) and \(T\) found, it is possible to find the position \(p_f\) and the orientation \(s_f\) of the currently analysed face.

\[
p_f = Rp_m + T
\]

\[
s_f = Rs_m + T
\]

Object Targeting
To know whenever the user is looking at an object, the system must first know where the table is in the environment’s point cloud. Thanks to its geometry, a table can be considered as a section of a plane and, therefore, the RANSAC algorithm be reused to detect the table. Since the Kinect camera must be on or near the table, it is possible to detect it by letting RANSAC only analyse points that are no more than \(2m\) away from the camera. There is also no need to use a sample bigger than 3 points, as a table is usually a smooth surface and, therefore, it is easier to detect accurately than a face. Since it is not pretended to move the table and the camera during the usage of this system, it is only required to detect the table one time.

Depending of the environment, the table may not be detected. If the user is close to a wall, it may consider it a “better” plane than the table, if it has more inliers. So, it may be required to run RANSAC multiple times, depending of the number of walls of the captured environment. Every new run of RANSAC must analyse again the point cloud, but without the inliers of the already detected planes. At the end, a sample of planes is collected. Thanks to the perspective of the camera, the plane of the table is the plane that looks similar to an horizontal plane. Thanks to this, the system can know which plane describes the table by checking the dot product of each normal vector with the vertical axis (\(y\) axis). With normalised vectors, the absolute value of the pretended plane will be the closest to 1.

Afterwards, the system must find, in every frame, the intersection between the table and the line of sight. The line of sight \((l)\) can be define as a line by using the parameters of the head pose, as shown by (??).

\[
l \equiv k.s_f + p_f, \quad k \in \mathbb{R}
\]

In order to find the intersection \((o)\) between the line of sight and the table, it is required to find the value of \(k\) that makes that point belong to both geometrical models \((k_o)\). The parameter \(t\) represents the normal vector of the table’s plane.

\[
\begin{cases}
    o = k_o.s_f + p_f \\
    t^T.o = -1
\end{cases} \quad \Rightarrow k_o = -\frac{t^T.p_f + 1}{t^T.p_f}
\]

By replacing \(k\) by \(k_o\) on equation (??), the intersection point is found. The final result of (??) has a singularity when \(t^T.p_f\) is equal to zero. In this situation the table’s vector is perpendicular to the line of sight’s vector. Therefore, is not possible to compute \(k_o\) in that situation and the system proceeds to analyse the next frame.

Since the table is considered a plane, the system believes that the table has no limits and may believe that this intersection is in a position out of bounds of the “real” table. This detail is not problematic as the objects’ location and identification is known before hand. Besides that, the system only detects any potential targeting if the intersection is near an object.

Results
3D Data
As mentioned before a Kinect camera was used to capture the 3D environment. Tests were made to qualitatively evaluate Kinect’s data \((\alpha = -0.0029\) and \(\beta_1 = 3.2154)\).

![Test images taken by Kinect](image)

Figure 4: Test images taken by Kinect

Figure 4 shows the type of images that are taken by the Kinect camera. Regarding the depth image, pixels with a darker blue are 3D points that are closer to the camera than points from pixels with lighter blue. Pixels in yellow have a value of 2047, which represent invalid depth values. Afterwards, both images were used to generate a 3D reconstruction of the environment, by following the strategy mentioned in section 3.1.

We also noticed that some colour pixels of the wall were placed in the point cloud of the user’s shoulder. This displacement most likely came from the intrinsic and extrinsic parameters. With this observation, another test was made to see how this displacement would affect the 3D placement of MTCNN’s keypoints. Figure 6 shows the results of this test. When looking at 6(b), the nose keypoint seems to be under it when it should be around the tip of the nose. However, it is not a serious error as each keypoint is still in its respective face feature’s region.

Face Detection
To compare the success rate of each detector, we photographed 4 users facing different directions.
For users P, M and I, they started the recording by looking at the camera. Then they looked at an object that was in front of them and performed the rest of the test by randomly looking at the table in order to change the head’s pose. User J’s recording came from another test and, therefore was recorded in a different environment. He started the recording by looking at the camera. Afterwards he looked at both his sides and then looked at the objects that were present on his table. His scenario is present in figure 4. The results of both detectors are shown in table 1. The total analysed frames for user P was 928, while user M had 840. User I and J had 1000 and 400 analysed frames respectively.

<table>
<thead>
<tr>
<th>User</th>
<th>Correct Detections</th>
<th>Failed Detections</th>
<th>False Detections</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>766 / 518</td>
<td>162 / 415</td>
<td>0 / 5</td>
</tr>
<tr>
<td>M</td>
<td>819 / 820</td>
<td>0 / 18</td>
<td>21 / 2</td>
</tr>
<tr>
<td>I</td>
<td>703 / 723</td>
<td>287 / 266</td>
<td>10 / 11</td>
</tr>
<tr>
<td>J</td>
<td>320 / 393</td>
<td>71 / 7</td>
<td>9 / 0</td>
</tr>
</tbody>
</table>

Table 1: Success rates of face detection (Viola-Jones / MTCNN)

As we can see, the success rates of both detector are similar. User P and, somewhat, user J showed a bigger difference. Viola-Jones and MTCNN had more correct detections with user P and J respectively. It is unlikely that the environment affected the success rate as users P, M and I where photographed in the same environment. But unlike the remaining users, P wore glasses during the test and has a white beard big enough to hide his whole neck. It is possible that this factors influenced the result. However, it is unknown to the writer of this thesis whether MTCNN had faces similar to P’s in its training dataset. User J also had a beard during the tests, but it was no bigger than 5 cm. The false positives where usually around the neck area. User J’s false detections from Viola-Jones were caused by a printer that was present in the background. Thus is recommended that this system works in a environment where there is minimal objects in the background and, of course with only one person in front of the camera.

Despite both detectors showed similar performance regarding face detection, MTCNN showed better results with face features detection and was chosen to be implemented on the system. Figure 8 shows a comparison on both detectors performance in detecting face features.

Face Tracking

When testing SIFT’s keypoint matching, we noticed that there were less matchings when the face looked away from the camera, as shown in figure 9. There were even some keypoints that were not correctly matched. At most, 20 match ups are found when the user is facing the camera.

There were frames where some keypoints would end up in an invalid depth pixel. With no depth information, these keypoints could not be transformed into 3D and, therefore these points could not be used to compute the new face pose. Since MTCNN and SIFT may gather around a collection of 20 matches, it would be unlikely to find less than 3 available matches, which is the minimum number of matches needed to find the rotation matrix.
Figure 8: Comparison between Viola-Jones and MTCNN face features’ detection

(a) Viola-Jones

(b) MTCNN

Figure 9: SIFT matching in different face angles. The images on the left represent the face model u, while the images on the right are the frames being currently analysed.

This scenario is more likely to happen when the user looks away, as SIFT may not find enough matches. If not enough matches are found, the system skips the current frame and starts to analyse the next one. A frame by frame analysis would increase the number of matches, instead of using a face model. However, this implementation is not robust, as any error that influences the pose’s result will carry over to any future frames.

We also checked how the pose behaved when updated with the computed transformation in each frame. We created an animation using the point clouds of each frame. We drew a by using the normal vector of the face to represent the line of sight. We noticed that this line presented a shaky behaviour as it would not stay still when the face was. This behaviour got worse when the user looked away from the camera.

Object Targeting

RANSAC runs 100 iterations in order to find a plane that fits in the point cloud of the table. It uses an error threshold of $10^{-4}$, as it filters the majority of the objects’ point clouds from becoming a plane inlier. The 3D points of the point cloud of the images in figure 4, that have a z coordinate smaller than 1.6 m, are used to test how well RANSAC detects the table. RANSAC was programmed to find 3 planes for this test in case that RANSAC detects other planes that has more inliers than the table. After 100 repetitions of this experiment, we noticed that the table was usually the second detected plane as it would first detect the side wall and the back wall was previously removed from the point cloud. In figure 10, the inliers of the first plane are represented in red, while the table is in green and the last plane is in blue on the body of the user.

For each repetition, the system checked the absolute value of the dot product between the vertical axis and the 3 acquired normal vectors in order to find the plane of the table from the list of collected planes. Figure 11 shows the output plane of one of the repetitions as well the normal vectors of the table found in each repetition.

However, during another test, we verified that the system could not correctly detect the table. Only a small section of the table had been captured by the camera and, since there were fewer points, it was harder for the system to consider them as points of a "horizontal" plane.

This finding led to a new set of experiments, in which we tested how much could object placement affect the detection of the table. We took photos when the table was empty (used as a reference table), with few objects, with some objects and when the table was completely covered with objects. The last situation was tested with objects laying horizontally on the table and also tested with tall ob-
Figure 11: Comparison between the plane and the points of the table

Figure 12: Colour image of the failed table test

Figure 13: Detected table of the failed table test (Front view)

Figure 14: Tested objects layouts

(a) With few objects
(b) With some objects
(c) Full of objects (horizontally)
(d) Full of objects (vertically)

We verified that the table detector can still viably detect the table with a considerable amount of objects (situation 14(b)) as if it was empty. However, the results were not as satisfactory when the table is fully covered in objects.

With horizontal objects, the detector considered that they "were" the table and classified some of their points as belonging in a plane. The last case showed a disastrous result, where the best plane match has an angle deviation of approximately 25° with the table plane.

Since the results of the face tracking problem have shown shakiness with the line of sight, it was not certain how close this intersection would be from the object of interest. Hence, new tests were performed where the user would fix his sight to a specific object. He started by looking at the camera and then fixed his gaze upon the object of interest. 15 shows the two tested situations.

Despite the shakiness, the intersection could stay, on average, less than 10 cm away from the mug, as shown in 16. Overall, the data showed a standard variation of 3.2 cm and it is likely that there is an offset error caused by the head’s positioning as the user was looking down, making it harder to have a clear view of the face.

The red bars represent distances bigger than 1...
Table 2: Relationship between the \( y \) axis and the computed normal vectors (dot product / degrees)

<table>
<thead>
<tr>
<th></th>
<th>Empty</th>
<th>Some</th>
<th>Lot</th>
<th>Full Horizontal</th>
<th>Full Vertical</th>
</tr>
</thead>
<tbody>
<tr>
<td>Worst</td>
<td>0.9776 / 12.15</td>
<td>0.9798 / 11.54</td>
<td>0.9718 / 13.64</td>
<td>0.9602 / 16.22</td>
<td>0.5446 / 57.00</td>
</tr>
<tr>
<td>Mean</td>
<td>0.9804 / 11.36</td>
<td>0.9834 / 10.45</td>
<td>0.9819 / 10.92</td>
<td>0.9762 / 12.53</td>
<td>0.7235 / 43.66</td>
</tr>
<tr>
<td>Median</td>
<td>0.9804 / 11.36</td>
<td>0.9834 / 10.45</td>
<td>0.9820 / 10.89</td>
<td>0.9757 / 12.66</td>
<td>0.7305 / 43.07</td>
</tr>
<tr>
<td>Best</td>
<td>0.9835 / 10.42</td>
<td>0.9879 / 8.92</td>
<td>0.9936 / 6.40</td>
<td>0.9913 / 7.56</td>
<td>0.8118 / 35.73</td>
</tr>
</tbody>
</table>

- The book did not suffer the same success as the intersection could not be closer than 30 cm, as shown in figure 17.

  We also tested two additional strategies to smoothen the shakiness of the line of sight. One uses the mean value of the 10 most recent poses of the face while the other has the only difference of using an weighted average, that is generated by a logarithmic function. In this last case, the most recent poses have a bigger weight than the oldest. These averages do not use poses modified by these strategies. Despite smoothing the shaky behaviour of the line of sight, these methods did not improve the distances.

  These tests were also performed with another face model. This face model was made of the images of the frame that was previous to the one used in these tests. Despite, being almost the same frame, this model created different results during the book test as there are more distances above 1 m, as shown in figure 18. Compared to the original results, this face model did not show different behaviour during the mug test.

Conclusions

During this project we developed an input system for a robotic arm. Its purpose was to provide an intuitive and simple method for robotic arm manipulation on objects by using its user’s head pose.

This work suggest that, despite the promising preliminary results, this system is still quite unpre-
dictable. The computed line of sight showed different behaviours with similar face models, when the user looks to her side. This situation is the most unstable as it is less likely to find correct keypoints matches. Therefore this system does not perform well with objects that are not in front of the user. They need to be in front of the user and around 20 to 30 cm away from each other, which suggests that the system should not handle more than 3 objects. The system also needs to be working in an environment with few objects in the background as the face detector could give a false positive. It is also ideal to have the user at around 1 m away from the camera, so that the colour images do not show a bad quality crop of the face.

It should be noted that these results did not come from real life tests. It was not possible to perform these tests, as MTCNNs Matlab implementation required hardware that was not possible to use during the development of this work. Hence, it is difficult to evaluate how different the results would be. Moreover, some members of the target audience showed great difficulty to stabilize their heads pose. So, even if better results were shown, it would be hard to conclude how reliable this system is, without testing it in the targeted audience.

For future work, it is recommended to test, in real life, how well this implementation can correctly detect when the user is looking at an object. It is possible that another detector must be used, but another option may not use the extra features found by MTCNN. A potential solution to the shakiness of the line of sight may be using more images when creating the face model. It may increase the number of correct keypoint matchings if the face model has more keypoints on its sides. It is also needed to check if it is required to add a time or frequency threshold to the intersection problem to avoid any mistake when choosing an object. This work was tested by assuming that the objects position is known. Therefore, an object detection algorithm or strategy needs to be developed if this work is to be carried out without giving the positions manually.

References