university of groningen / faculty of science and engineering

# Object Anchoring for Human-Robot Interaction: Connecting Sensor Data to Symbols

Bachelor's Project Thesis

Nikolai Herrmann, n.a.herrmann@student.rug.nl
Supervisor: Dr. Hamidreza Kasaei, hamidreza.kasaei@rug.nl
Department of Artificial Intelligence

**Abstract:** Tracking algorithms, which track objects in motion, tend to fail when objects get close to one another, overlap, or are temporarily occluded. To combat this limitation, the anchoring framework can be applied, allowing objects to be tracked symbolically such that they can be uniquely identified at any moment. This is achieved by maintaining correspondence between raw perceptual data from sensors and abstract symbols, using a matching function. Here, we use a bottom-up approach where the matching function either acquires a new object or reacquires a previously seen one. Four different binary classifiers were trained to accomplish this task. To avoid manual labelling we track objects separately by color to maintain a ground truth. In addition, two approaches of the reacquire functionality are explored: one which continuously anchors and one which anchors only stationary objects while still observing moving ones. Scenarios of different difficulties and category were tested including the Three-Card Monte and the Shell Game. Our results demonstrate limited success with the first approach due to data sensitivity, but the second approach shows clear improvements with the help of motion analysis and the Kalman filter.

## 1 Introduction

Three-Card Monte is a simple card game, often played on the street with bets. A player will initially be presented with three standard downward facing playing cards, one of which the dealer will flip to reveal its type. The target card is then flipped back and the dealer begins to shuffle the cards, switching around two cards at a time. The task of the player is to follow the initial flipped card and identify it again once the shuffling has stopped. The game appears straightforward at first but, in reality, it is not so simple. The Metropolitan Police Department in DC has a dedicated web-page explaining that Three-Card Monte is a scam due to its unforeseeable difficulty. They recommend to avoid the situation and to get the attention of a police unit if approached to play. A similar difficulty occurs in the Shell game presented in the work of Persson et al. (2019), where cups and a small ball hidden inside one of the cups are used instead of cards. These scenarios raise the following questions:

- *What makes these games so difficult?*
- *How can we build an autonomous system which can cope with the task of playing such games?*

The introduction of these questions not only stems from curiosity but possible approaches to solutions have great applications in navigation and surveillance domains (Coradeschi & Saffiotti, 2000). Coradeschi et al. (2013) present examples, from unmanned aerial vehicles collaborating with one another to robots mastering spatial relations. These relations allow for a high-level human-robot interaction. For example, the robot would be able to handle the following query: "pick up the red book on the counter".

We will start by examining the first question more formally with the hope that it will lead us to the second. The first thing to consider is that despite all three cards being identical when face down, the target card is most valuable. Thus, the player holds an abstract representation of the cards no matter their location, rotation or whether they are flipped or not. While shuffling, the player is continuously provided with perceptual information. The difficulty of the game arises from having the task of

constantly matching perceptual data of a physical card with its corresponding abstract representation (Coradeschi & Saffiotti, 2000). For example, if the target card is swapped with the middle laying card, the player needs to now match the middle card with the target card. Saffiotti (1994) was one of the first to define the process of maintaining such a correspondence as *anchoring*.

This leads us to the answer of the second question, that to build an autonomous system capable of successfully playing Three-Card Monte, an anchoring system is needed. It is important to note that the accuracy of an anchoring system is limited by its matching capability (Coradeschi & Saffiotti, 2000). Bredeche et al. (2003) report that designing and programming a matching function based on rules is a difficult task because of constant changes in the environment. Therefore they recommend a *machine learning* approach. This is achieved by supervised learning where the task is to learn a function that decides whether incoming perceptual data matches with a specific abstract representation. A model is trained on snapshots of the environment where the correct matching decisions are known (Bredeche et al., 2003).

Persson et al. (2019) successfully trained an accurate anchoring system for the Shell game. Their approach is unique, as they focused on coupling their anchoring system with probabilistic reasoning in order to establish a semantic world model. This type of modelling is based on the works of Elfring et al. (2013) which demonstrated the workings of multiple hypotheses anchoring to make predictions about object entities. Using this method is necessary as the intricacies of building an anchoring system truly erupt when it becomes a requirement to accurately maintain objects which are fully occluded. This is especially the case with the Shell game where the ball is underneath one of the cups for the majority of the game.

### 1.1 Our Approach

In order to train a model for an anchoring system, a ground truth must be maintained. In terms of anchoring this simply means that at any given frame (i.e. snapshot of the environment) it is known whether any detected objects have been seen before or are new ones. This is easily achieved by assigning each object an unique identification (ID). The methodology used by Persson et al. (2019) to accomplish this is cumbersome as they manually labeled each object in each frame.

Alternative to manual labeling, it is possible to track objects by some unique feature. This can be as simple as detecting the type of object. For example, a convolutional neural network (CNN) such as YOLO could be trained to detect specific objects in real-time (Redmon & Farhadi, 2018). Moreover, symbols are also an option; objects would be marked by a character which maps to a unique ID. Our solution was to maintain a ground truth of objects using *color*. The advantage of color is that in the case an object is partially occluded, it will still be detectable. In this way, the process of collecting training data from a given input stream can be automated. For this method to be successful for training, we make the core assumption that each object in the environment has a unique and differentiable color.

Using the collected training data, we evaluate the accuracy of different classifiers comparing: Naive Bayes, K-Nearest Neighbour (KNN), Logistic Regression and Decision Trees. Naive Bayes and Logistic Regression are both linear models making classifications based on probability (Bhowmik, 2015). KNN and Decision Trees in contrast are non-linear using distance to determine similarity and a hierarchical structure respectively (Mohanapriya & Lekha, 2018). Using the introduced color tracking approach we investigate:

*How accurately can a classifier (matching function) be learned to maintain anchors of physical objects and which type of classifier does this best?*

The exact notion of the implemented anchoring framework is presented in the following section.

## 2 Framework Specification

We here adopt the formal anchoring definitions proposed by Coradeschi & Saffiotti (2000). They introduce the following two time invariant sub-systems summarised in Figure 2.1, where the following definitions apply:

- **Symbol system** — manages the abstract representation of objects and contains two

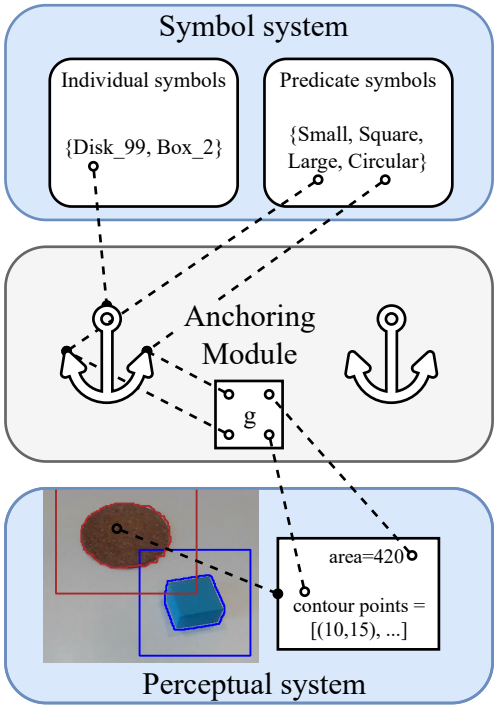


**Figure 2.1: Showing the three main components of an anchoring system based on Figure 1 and 2 presented in Coradeschi & Saffiotti (2003).**

sets. The first is a set of individual symbols: $\mathcal{X} = \{x_1, x_2, ...\}$. Each element uniquely identifies a physical object in the environment as shown in Figure 2.1 where we have: $\mathcal{X} = \{\text{Disk\_99}, \text{Box\_2}\}$. The second set, containing predicate symbols, is used to describe individual symbols: $\mathcal{P} = \{p_1, p_2, ...\}$. Disk_99 for example is defined by the set $\{\text{Large}, \text{Circular}\}$.

- **Perceptual system** — regulates incoming perceptual data. Similarly as above, two sets are included in the system. Firstly, it includes a set of percepts: $\Pi = \{\pi_1, \pi_2, ...\}$. In terms of video input, a percept is a pixel region indicating the detection of physical object. For example, in Figure 2.1 the blue and brown outlining contours within the bounding boxes indicate percepts. From each percept measurable values can be extracted, known as attributes. $\Phi = \{\phi_1, \phi_2, ...\}$ defines the set of all attributes. In Figure 2.1 we have $\Phi = \{shape, size\}$. Each attribute has values in a given domain $D_i$. It should be noted that color cannot be an attribute as it is used as a ground truth reference.

The task of the anchoring system is to then make a correspondence between individual symbols and percepts with the help of a *predicate grounding relation* ($g$) (Coradeschi & Saffiotti, 2003). We will characterize $g$ as a matching function. In order to implement such a correspondence between these two systems an internal data structure known as an *anchor* is used (Coradeschi & Saffiotti, 2000). An anchor is denoted by $\alpha_t^x$, such that at each time step, $t$, it identifies itself with exactly one individual symbol, $x$ (Persson et al., 2019). Moreover, Persson et al. (2019) define an anchor that has matched with a given percept to be in a *grounded* state.

The general process of correspondence can be achieved through a bottom-up, top-down or hybrid approach (Coradeschi & Saffiotti, 2003). We will use bottom-up anchoring as recommended by Persson et al. (2019), whereby environment interactions apparent in sensor data initiate anchor creation. The standard approach in the anchoring framework is for $g$ to map between discrete predicate symbols and continuous attributes. Persson et al. (2019), however, highlight that this may neglect rich information provided by the perceptual system. Therefore, they propose to move the matching function down to the perceptual system. Correspondence is then achieved through two core functions, given a newly detected physical object:

- **Acquire** — is called to create a new anchor $\alpha$. This means that no existing anchor matches the detected object. The base functionality is to allocate for a new structure $\alpha$: initialized with a unique symbol, $x$, time step, $t$, and the attributes, $\Phi$, from the perceptual data.
- **Reacquire** — is called to update the anchor, $\alpha$, when a match has been found. The perceptual data from time step $t - k$ will be updated to the most recent attribute measurements extracted at $t$.

The functionality incorporating a bottom-up approach can be seen in the revised Figure 2.2.

In addition to these two functions, it is custom in the anchoring framework to have a special case of the *reacquire* function. Coradeschi & Saffiotti (2000) introduced this functionality with a *track*

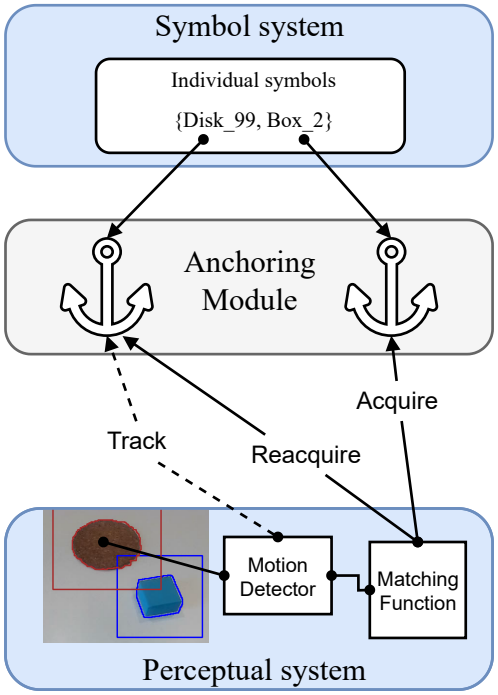


**Figure 2.2: Showing anchoring functionalities: *acquire*, *reacquire* and the special case *track*. Based on Figure 2 illustrated in Persson et al. (2019).**

function, shown in Figure 2.2 by the dotted line. The idea behind this addition is that if an object is constantly being observed it may be redundant and thus computationally expensive to repeatedly reacquire it. Persson et al. (2019) note that this function has under gone much revision; the latest improvements include the conjunction of a point cloud library presented by Persson et al. (2017). Since there is no standardized approach to a track function, the upcoming methodology will be divided into two approaches. The first approach will attempt to build a matching function without the need of supplementary tracking while the second approach will incorporate a track function based on motion detection as shown in Figure 2.2.

# 3 Methods

All code was written in Python (v3.8.1) using the OpenCV library (v4.5.5) for computer vision and the scikit-learn (v1.0.2) package for machine learning models (Bradski, 2000; Pedregosa et al., 2011). The complete code and other minor dependencies can be found on Github under github.com/NikolaiHerrmann/Anchor-Tracking. General methodologies which apply to both approaches are described first.

## 3.1 Color Segmentation

### 3.1.1 Color Space

The standard color space for images and videos is RGB (red, green, blue) as it is most suitable for the majority of current displays. However, a caveat of RGB is that different levels of brightness results in shifting RGB values (Sebastian et al., 2008). Thus, continuous frames in a video may be extremely susceptible to color instability. RGB can be converted to other color spaces to overcome this limitation. Sebastian et al. (2008) found that tracking accuracy was improved when using YCbCr or HSV (hue, saturation, value) rather than RGB or grayscale.

The HSV color space was selected. It is part of the Munsell-like color spaces which allows humans to pick colors more easily (Bradski, 2000). The hue channel dictates the color type, which can be fine-tuned through the saturation channel, controlling the amount of color used, and the value channel which regulates the brightness of the color. A graphical user interface (GUI) was built to segment an objects color by manually narrowing down the threshold of each HSV channel. This can be seen in Figure 3.1 where the minimum and maximum HSV channel bounds for the color light blue are determined.

### 3.1.2 Contours

The thresholding results in a binary image mask; an array of intensities of 0 (black) indicating the background or 255 (white) marking an object. A morphological operation is then applied such that the mask is dilated by a kernel of size $3 \times 3$ in order to combat incorrect thresholds at the edges due to shadows. This mask is then used to find contours: an array of continuous points that define the boundary of a (white) shape (Bradski, 2000). OpenCV uses a border following algorithm to make topological analyses such as contouring (Suzuki & Abe, 1985). The contour is then used to calculate

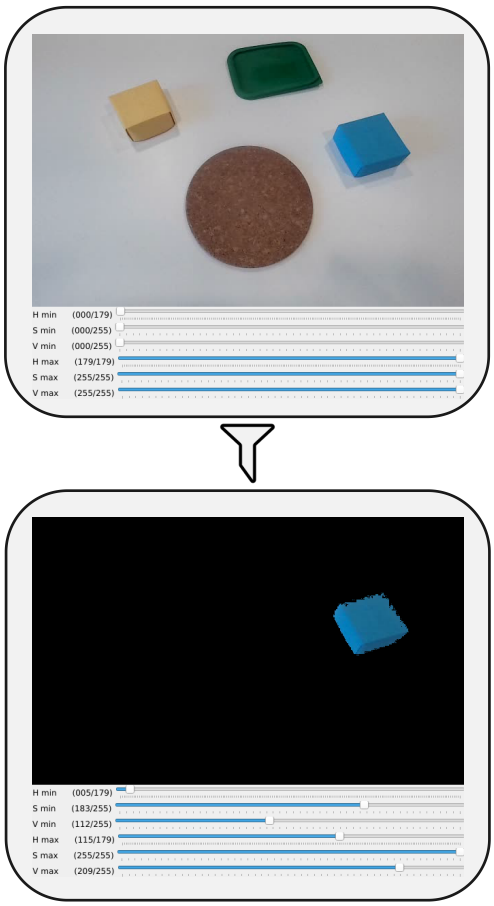


**Figure 3.1: GUI showing how changing HSV bounds can mask a specific color. The final lower and upper bounds for the light blue box are:** $[5, 183, 122]$ **and** $[115, 255, 209]$**. The bottom image is produced by a bitwise AND gate between the original image and the mask.**

$(j, i)$ - image moments where $i$ and $j$ denote the order as seen in Equation 3.1 (Rocha et al., 2002).

$$M_{ji} = \sum_x \sum_y (\text{array}(x, y) \cdot x^j \cdot y^i) \tag{3.1}$$

Here $x, y$ pairs are point coordinates and `array` gives the corresponding pixel intensity, resulting in a weighted average. From this we have the following properties for binary images using the $0^{th}$ and $1^{st}$ order moments:

$$\text{area} = M_{00},\ \bar{x} = \frac{M_{10}}{M_{00}},\ \bar{y} = \frac{M_{01}}{M_{00}} \tag{3.2}$$

where $\bar{x}$ and $\bar{y}$ denote the centroid coordinates of the contour (Rocha et al., 2002). Area is the pixel count of the object. This gives two core attributes of a detected object: its position and size.

Occasionally, it may occur that OpenCV finds multiple contours due to slight threshold inaccuracies; in those cases the contour with the maximum area is chosen.

### 3.1.3 Object Materials

For the HSV color segmentation to be successfully it is important for the objects to be uniform in color and have matte surfaces to reduce specular highlights. The utilized objects were thus made out of either paper, rough plastic or cork.

## 3.2 Matching Function

The general anchoring is the same for both implemented approaches. Given a detected percept $\pi$, the pseudocode presented in Procedure 3.1 will show how to ground it.

The `matching_distance_vals` function, presented in Procedure 3.1, needs to compare the set $\Phi_y$ to the set $\Phi_x$. This is done by comparing corresponding attributes $\phi_y^{attr}$ and $\phi_x^{attr}$ individually. Persson et al. (2019) illustrate this further by introducing a matching distance value $d$ for each attribute:

$$d_{x,y}^{attr}(\phi_y^{attr}, \phi_x^{attr}) \tag{3.3}$$

In other words, the task of the matching function combines the matching distance values for each attribute to make a binary classification of whether $\Phi_x$ and $\Phi_y$ represent the same physical object. This will be achieved by having a model $\hat{f}$ estimate whether there is match given $K$ attributes:

$$\hat{f} : \mathbb{R}^K \to \mathbb{R} \tag{3.4}$$

The models mentioned in section 1.1 have been selected for their minimal hyperparameters to reduce additional complexity. However, which attributes are utilized and how they are updated with `reacquire` depend on the approach.

**Procedure 3.1** Anchoring of percept $\pi_y$. *Symbols subscript by $y$ refer to an unknown object while symbols subscripted with $x$ belong to an existing anchor. $k$ denotes a unique past time for an anchor, it will always be less than $t$. Unless stated otherwise, $T$ is set to the default* 0.5

**Input:** percept $\pi_y$, set of all anchors A, time step $t$ and matching threshold $T$.

$\Phi_y \leftarrow$ extract_attributes($\pi_y$) # features as set
*lookup_table* $\leftarrow$ empty_lookup_table()
**for all** $\alpha_{t-k} \in$ A **do**
  $\Phi_x \leftarrow$ get_attributes($\alpha_{t-k}$)
  $features \leftarrow$ matching_distance_vals($\Phi_x, \Phi_y$)
  $prob \leftarrow$ classifier_predict($features$)
  **if** $prob >$ T **then**
    *lookup_table*$[\alpha_{t-k}] \leftarrow p$
  **end if**
**end for**
**if** is_empty(*lookup_table*) **then**
  acquire(A, $\Phi_y, t$)  # add
**else**
  $\alpha_{t-k} \leftarrow$ max(*lookup_table*)
  reacquire($\alpha_{t-k}, \Phi_y, t$)  # update
**end if**

### 3.2.1 Approach 1: Framewise Anchoring

In this approach the system attempts to detect objects in every frame. If an object is found, Procedure 3.1 is called with the detected object as the input.

An object is identified by where it is located and its exterior shape. The first two attributes, are thus, position ($\phi_y^{pos}$) and size ($\phi_y^{size}$), calculated using Equations 3.1 and 3.2 respectively. Moreover, the objects rotation ($\phi_y^{rot}$) can be determined by finding the smallest rotated rectangle as shown in Figure 3.2.

Another distinct feature is if the object is moving by some external force. A simple way to detect movement is by calculating the difference in position. This can be achieved by maintaining a buffer of past positions in order to get a position measured $n$ frames back; here, $n$ was chosen to be 10 based on small tests. The difference between the current position $p$ and an old position $q$ can be seen as a vector difference: $k = p - q$. The magnitude $|k|$

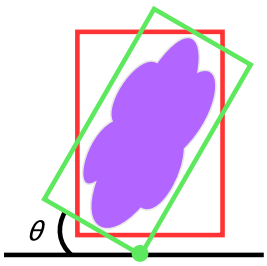


**Figure 3.2: The green bounding box shows the minimum (rotated) sized rectangle while the red bounding box indicates the minimum up-right rectangle. $\theta$ is always calculated from the lowest point parallel to the red bounding box and will range from $[0, 90]$.**

can be calculated providing an indication of speed, while the unit vector $\hat{k}$ is used as a indicator of direction. As a result, we add the attributes $\phi_y^{mag}$ and $\phi_y^{dir}$ to $\Phi_y$ resulting in the set:

$$\Phi_y = \{\phi_y^{pos}, \phi_y^{size}, \phi_y^{rot}, \phi_y^{mag}, \phi_y^{dir}\} \tag{3.5}$$

In addition to selecting attributes, it must be determined how the matching distance values are calculated for each attribute. For the position attribute the matching distance value is determined by finding the Euclidean distance between the previous position $\phi_x^{pos}$ and the current position $\phi_y^{pos}$:

$$d_{x,y}^{pos}(\phi_y^{pos}, \phi_x^{pos}) = L^2(\phi_y^{pos}, \phi_x^{pos}) \tag{3.6}$$

Directions are matched by a dot product:

$$d_{x,y}^{dir}(\phi_y^{dir}, \phi_x^{dir}) = \phi_y^{dir} \cdot \phi_x^{dir} \tag{3.7}$$

For example, if the two vectors are parallel their dot product is 1. The remaining attributes (area, magnitude and rotation) are calculated by simply taking an absolute difference:

$$d_{x,y}^{attr}(\phi_y^{attr}, \phi_x^{attr}) = \text{abs}(\phi_y^{attr} - \phi_x^{attr}) \tag{3.8}$$

### 3.2.2 Approach 2: Anchor with Tracking

The idea behind the second approach is that instead of anchoring moving objects by attempting

to quantify the motion through rotation, magnitude and direction attributes we track moving objects separately. To detect moving objects a KNN background subtractor, which subtracts the current frame from a past background frame (Zivkovic & Van Der Heijden, 2006), is used. A difference of 20 frames was selected. This allows for a mask to be calculated indicating the pixels of the objects in motion as shown in Figure 3.3.

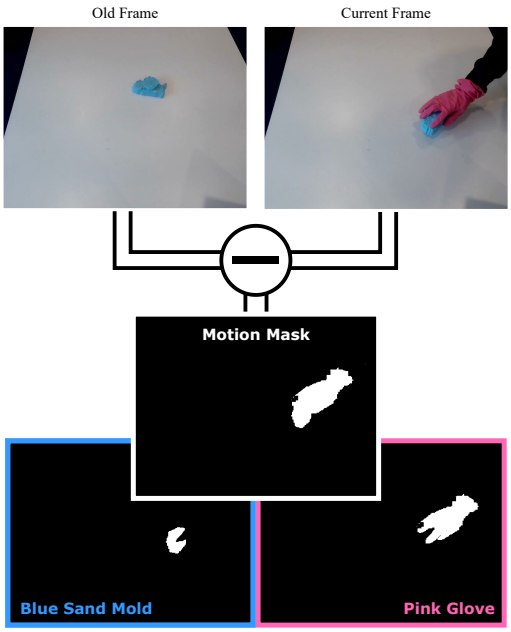


**Figure 3.3: Background subtraction**

With Procedure 3.2 it can be tested if a specific contour, such as the blue sand mold or the pink glove are in motion.

**Procedure 3.2** Check if a contour mask is in motion. *The variable motion_thresh was set to 100.*

1: $test\_mask \leftarrow contour\_mask \;\&\; motion\_mask$
2: $in\_motion \leftarrow (\text{count\_nonzero}(test\_mask) > motion\_thresh)$

If an object is in motion we prevent it from being anchored and instead update the position attribute of the anchor it was last grounded to. Additionally, the position coordinates are inserted into a Kalman filter. This is a simple strategy in order to make predictions about hidden objects. This filter was first introduced by Kalman (1960) and allows for the estimation of linear systems. As a result, if an object is not detected its position is predicted by the filter. Naturally if an object is off screen updating the position has little benefit and those updates are ignored. The implementation follows a standard 2D tracking setup detailed by Sadli (2022). The following parameters were applied:

| Parameter | Description | Value |
|---|---|---|
| $\Delta t$ | period ($\frac{1}{\text{FPS}}$) | $\frac{1}{30}$ |
| $u_x$ | acceleration along $x$ | 1 |
| $u_y$ | acceleration along $y$ | 1 |
| $\sigma_a$ | noise magnitude | 5 |
| $\sigma_x$ | SD of $x$ measurement | $\frac{1}{1000}$ |
| $\sigma_y$ | SD of $y$ measurement | $\frac{1}{1000}$ |

**Table 3.1: Kalman filter parameters, where SD refers to standard deviation and FPS to frames per second. Parameters are adapted from Sadli (2022).**

## 3.3 Data Collection

To train a model using supervised machine learning labeled data is required. To do this, videos were created showing colored objects being moved around on a table by human hands. Each object had a unique *color* ID. Procedure 3.1 can be adjusted, as shown in Procedure 3.3, such that labeled data can be generated.

**Procedure 3.3** Readjustments of Procedure 3.1 to collect data of correct matches.

**Input:** percept $\pi_y$, set of all anchors A and time step $t$

1: $color\_y \leftarrow \text{extract\_color}(\pi_y)$
2: $\Phi_y \leftarrow \text{extract\_attributes}(\pi_y)$
3: **for all** $\alpha_{t-k} \in$ A **do**
4: $\quad color\_x \leftarrow \text{get\_color}(\alpha_{t-k})$
5: $\quad \Phi_x \leftarrow \text{get\_attributes}(\alpha_{t-k})$
6: $\quad features \leftarrow \text{matching\_distance\_vals}(\Phi_x, \Phi_y)$
7: $\quad target \leftarrow (color\_x == color\_y)$
8: $\quad \text{write\_to\_file}(features, target)$ # save data
9: **end for**
10: update(A, $\Phi_y$, $color\_y$, t) # acquire/re-acquire

Here `update` checks if $\Phi_y$ is already in A by comparing the colors of all anchors. If an anchor ($\alpha$) with $color_y$ is found, its attributes are updated to $\Phi_y$, otherwise a new anchor is created.

All videos were recorded with a Aukey PC-LM1 1080p webcam sampled at 30 FPS and down-scaled to a $640 \times 480$ resolution. This is OpenCV's default resolution and allows for efficient real time viewing on basic hardware. During all videos dishwashing gloves were worn, as shown in Figure 3.3. This prevented any similarities in HSV bounds between the skin of the manipulating hands and the objects.

Moving, stacking, disappearance were three different testing scenarios created in order to test the robustness of the two anchoring systems.

- ***Moving*** was used to test basic functionality as it included videos of objects being moved around the environment with minimal occlusion.
- ***Stacking*** targeted occlusion as objects were piled on top of each other in various manners.
- ***Disappearance*** contained videos were objects where not visible for a few seconds. This included videos of the Three-Card Monte game and the Shell game.

### 3.4 Classifier Parameters

The KNN classifier was trained with $k = 3$. Odd numbers of $k$ are typical for binary classifiers and low numbers are less computationally expensive. A max depth of 10 was set for the decision tree to prevent overfitting and to reduce computational complexity. The implementations of Naive Bayes and Logistic Regression provided by the *scikit-learn* package require no parameters to be set.

### 3.5 Data Split

36 videos varying between 30 seconds to 1.5 minutes in length were made. Half of the videos were used for training and the other half for testing. This generous split was reasoned by the fact the classifiers had a lower number of input features. The upcoming learning curves in section 4 will confirm this. Both testing and training received 6 videos from each of the scenarios introduced above.

### 3.6 Preprocessing

An important byproduct of Procedure 3.3 is that as the set of anchors A grows, more observations of where *target* is `false` will be saved. This results in an imbalanced dataset with many more observations showing matching distance value thresholds that do not match. A consequence of the classifier being trained on a class imbalanced dataset is that it may form a bias towards the majority class (Kotsiantis et al., 2006). A simple solution is to randomly undersample the majority class. Kotsiantis et al. (2006) note that a major drawback of undersampling is the loss of potentially useful data which could improve the classifier. We neglect this imperfection as our data is not scarce.

### 3.7 Evaluation

To evaluate our anchoring system, the output of function `classifier_predict` presented in Procedure 3.1 needs to be monitored. As the set of anchors grows, a lot of true negatives (TN) will naturally occur. However, it is more important that the system picks exactly the right anchor (if an acquire is possible), than declining the majority of incorrect anchors. It is therefore unwise to take any metric using TN such as accuracy. Instead we will use precision and recall as performance metrics being defined as follows:

$$precision = \frac{TP}{TP + FP},\ recall = \frac{TP}{TP + FN} \tag{3.9}$$

Furthermore, since neither can be maximised simultaneously, the "harmonic" mean of the two will be used. This is encapsulated by the $F_1$ score as follows:

$$F_1\ score = \frac{2 \times precision \times recall}{precision + recall} \tag{3.10}$$

## 4 Results and Discussion

### 4.1 Training

To verify whether enough data was collected, learning curves can be plotted as shown in Figures 4.1 and 4.2 for each approach respectively. This was

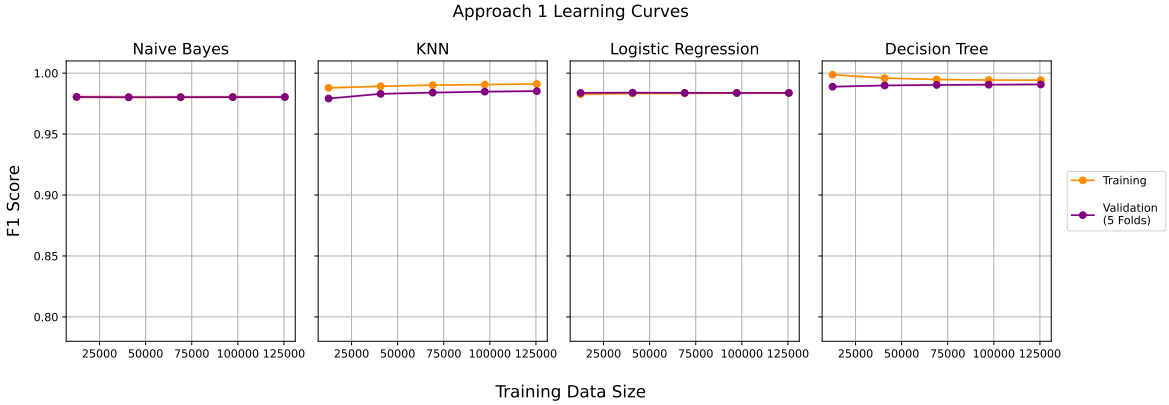


**Figure 4.1: Classifier learning curves with increasing number of observations for approach 1.**

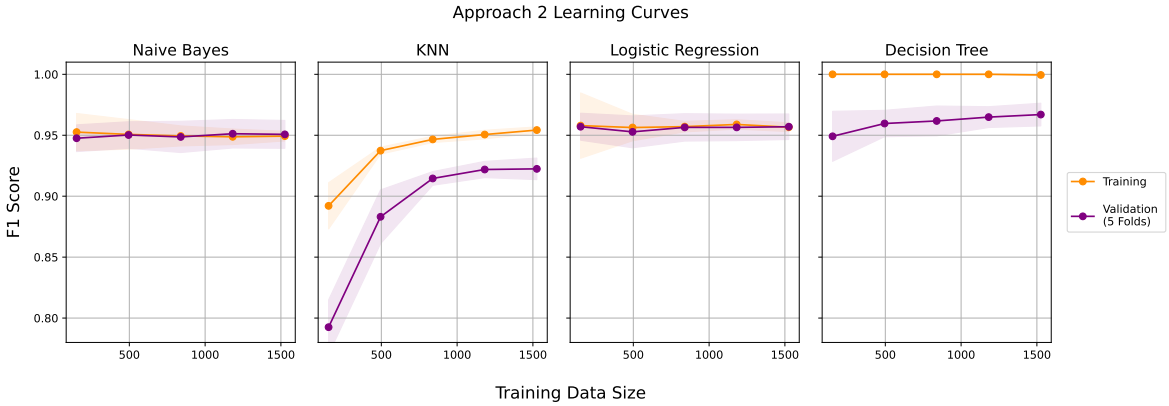


**Figure 4.2: Classifier learning curves with increasing number of observations for approach 2.**

done by training each classifier 5 times with increasing amount of observations. For each training, 5-fold cross validation was applied.

For approach 1 it can be clearly seen that enough data was collected, as there is an almost perfect convergence between the training and validation lines while showing high $F_1$ scores at around 0.97. Approach 2 uses the same videos but does not do frame-wise classification and hence has drastically less observations in comparison. Yet, similar results can be seen with slightly lower $F_1$ scores just above 0.95 for the majority of classifiers. For KNN and the Decision Tree, additional observations may yield better performance.

## 4.2 Testing

For each classifier, a model was trained and then evaluated on the three testing scenarios. This was done by first calculating the average recall and precision per scenario (see Appendix A). From these, an average $F_1$ score was calculated which was then averaged across approaches and testing scenarios.

Figure 4.4 shows a comparison of two approaches illustrating that all classifiers perform better using approach 2.

Approach 2's improvements can also be inspected visually as shown in Figure 4.3. It can be seen that as the coaster is being pushed towards the top of the table approach 1 classifies it as a new object three times. Approach 2 in comparison simply tracks the position maintaining the same ID.

Furthermore, in Figure 4.4 We see almost identical average performances for Decision Tree, Logistic

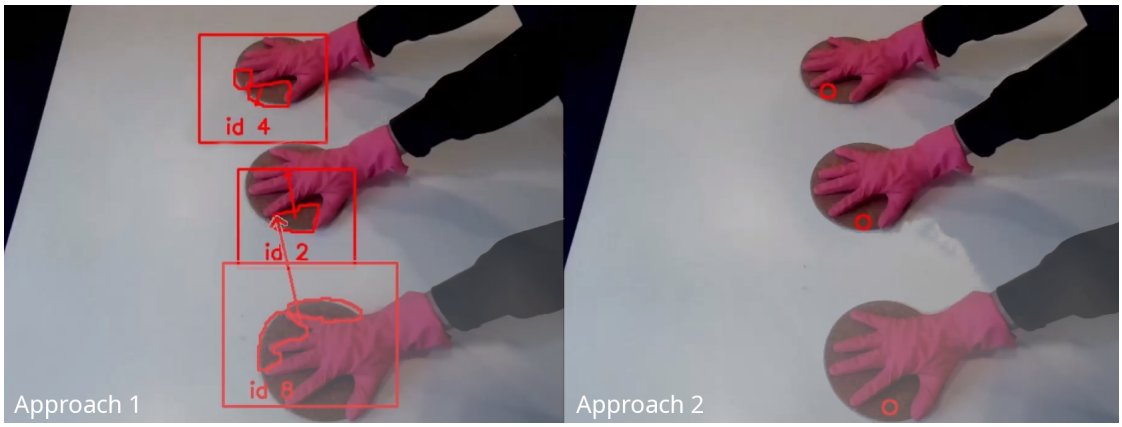


**Figure 4.3: Comparing approaches 1 and 2 on a *moving* scenario video. Images depict hands from three different frames, as the brown coaster is being moved from the bottom of the table to the top.**

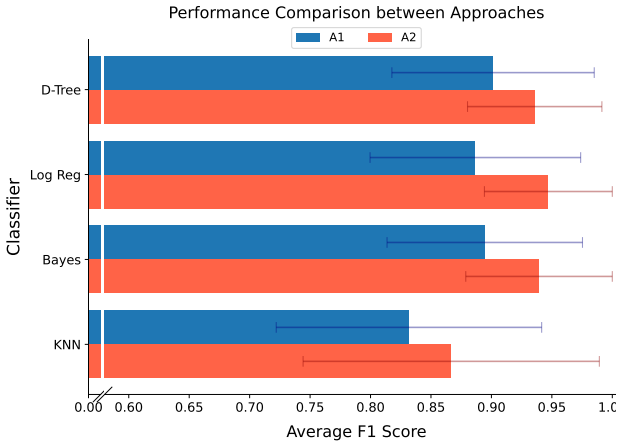


**Figure 4.4: Showing classifier performance for each approach where *A1* and *A2* denote approach 1 and 2 respectively. Error-bars show $\pm$ 1 *SD*.**

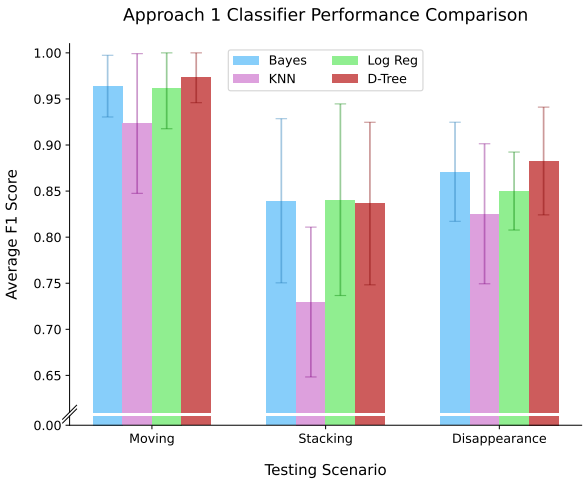


**Figure 4.5: Showing classifier performance for approach 1. Error-bars show $\pm$ 1 *SD*.**

Regression and Bayes between the two approaches.

### 4.2.1 Approach 1

Additionally, classifier performance can be evaluated across the three testing scenarios as shown by Figures 4.5 and 4.9.

In Figure 4.5 it can seen that for Approach 1 the classifiers perform best in the *moving* scenario, all reaching an $F_1$ score higher than 0.90.

*Stacking* is much more difficult for the frame-wise classification; more errors occur pushing all $F_1$ scores below 0.85. Moreover, it was surprising to see KNN drop below 0.75. Yet, these scores highly fluctuate as shown by the large error-bars. Logistic Regression, for example, has a standard deviation of 0.10. Lastly *disappearance* preforms slightly better compared to *stacking* with Naive Bayes and the Decision Tree getting a score above 0.85 but still below 0.90.

In Figure 4.5 there is a suggestion that Decision Tree generally performs the best. To test the null

hypothesis of whether all classifiers perform equally we follow the method presented by Demšar (2006) and undergo a Friedman test; a non-parametric repeated-measures ANOVA to compare classifiers across the same data sets. Here a data set refers to one of the 18 testing videos. For approach 1, a Friedman test showed a significant difference between classifiers, $\chi^2_F(3) = 26.2, p < 0.01$. A post-hoc test, shown in 4.1, reveals that only KNN performs significantly worse compared to the other three.

| | Bayes | KNN | Log Reg |
|---|---|---|---|
| KNN | $< 0.01$ | - | - |
| Log Reg | 1.00 | 0.021 | - |
| D-Tree | 0.891 | $< 0.01$ | 0.407 |

**Table 4.1: Showing $p$-values using Conover's all-pairs test with Bonferroni correction for approach 1. Significance is highlighted in gray.**

Lastly, it should be checked which features are contributing the most to a classification decision. The *scikit-learn* package automatically calculates feature importance for decision tree given its default hierarchical setup. This is done by calculating the mean decrease in impurity (MDI) for a tree (Pedregosa et al., 2011) as shown in Figure 4.6. A higher value implies greater importance.

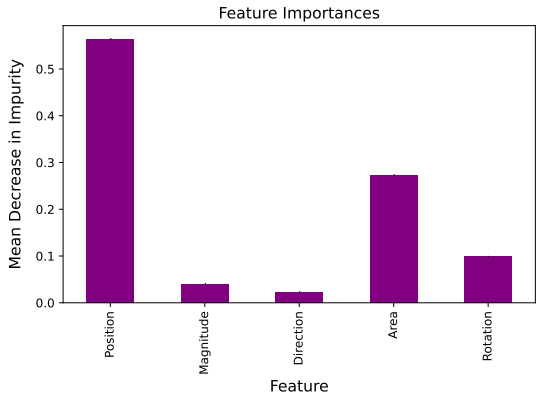


**Figure 4.6: Showing feature importances for Decision Tree.**

We see the highest decrease with the position attribute followed by area. This is expected, as no match should be made for objects far apart. Magnitude and direction in comparison are extremely low. To check whether they actually improve the model, models would have to re-trained, holding out both these variables and one at a time.

### 4.2.2 Approach 2

For approach 2 we observe higher $F_1$ scores for each scenario, as shown in Figure 4.9, compared to approach 1.

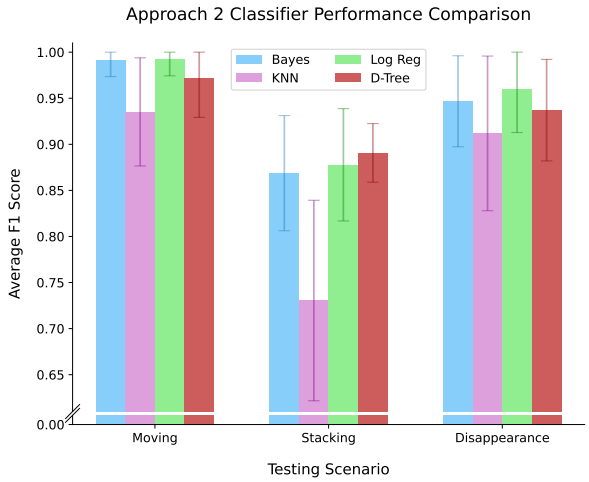


**Figure 4.9: Showing classifier performance for approach 2. Error-bars show $\pm$ 1 $SD$.**

For example, Bayes and Logistic Regression increase by 0.03 in both the *moving* and *stacking* scenarios. For the *disappearance* scenario all classifiers drastically improve, with Logistic Regression rising by 0.10. The anchoring system still has the most trouble with *stacking*.

A Friedman test for approach 2 also showed a significant difference, $\chi^2_F(3) = 18.6, p < 0.01$. A post-hoc test again reveals that KNN has significantly lower performance compared to Naive Bayes and Logistic Regression as shown in Table 4.2.

| | Bayes | KNN | Log Reg |
|---|---|---|---|
| KNN | 0.019 | - | - |
| Log Reg | 1.00 | $< 0.01$ | - |
| D-Tree | 1.00 | 0.074 | 0.817 |

**Table 4.2: Showing $p$-values using Conover's all-pairs test with Bonferroni correction for approach 2. Significance is highlighted in gray.**

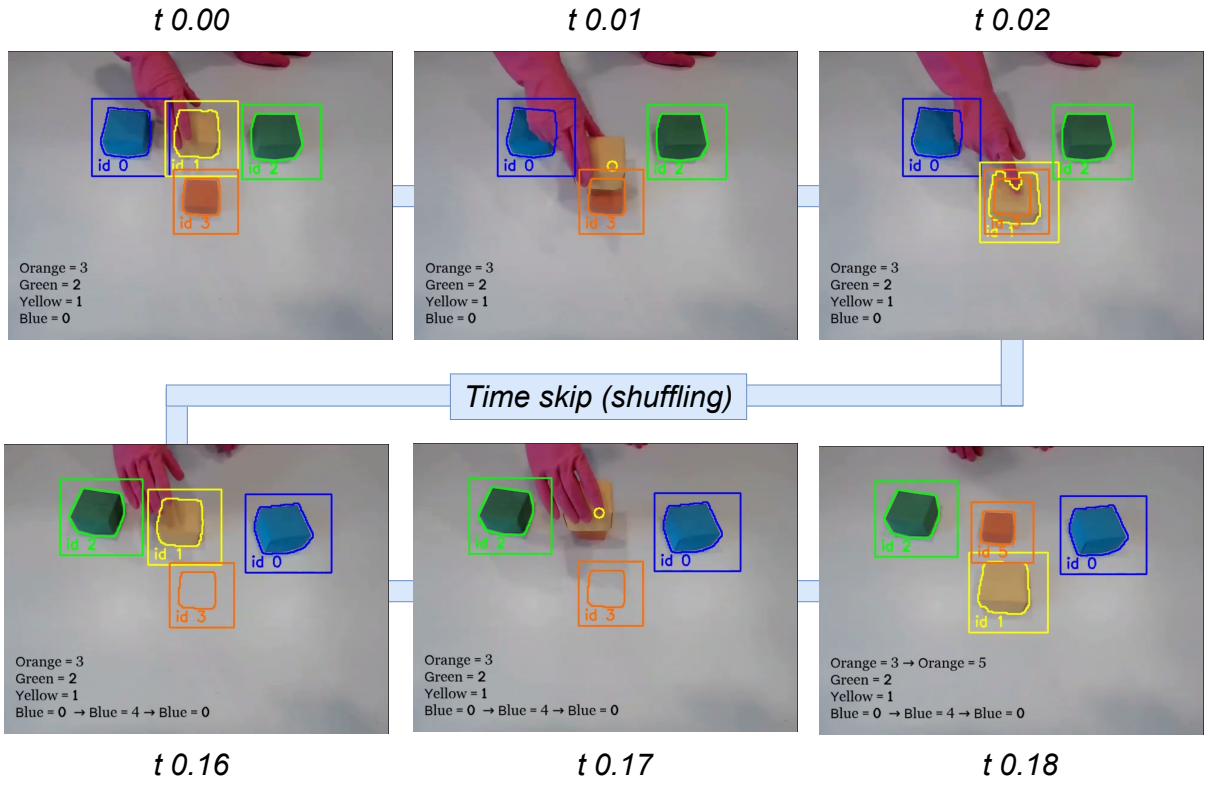


**Figure 4.7: Showing a test example where the shell game is played. $t$ is in seconds.**

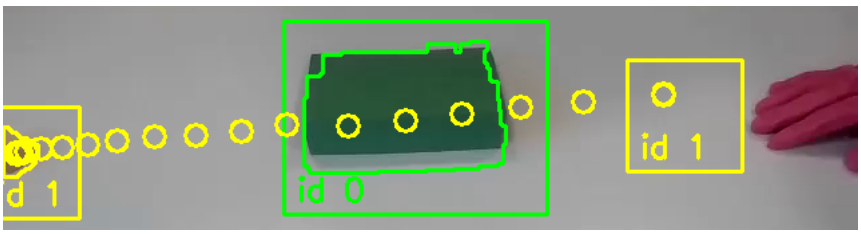


**Figure 4.8: A yellow paper box is pushed underneath a green box from right to left. Circles indicate the presence of the track function.**

Interestingly, Decision Tree just falls short of a significant $p$-value ($< 0.05$) indicating that it is not the best classifier as in approach 1. Logistic Regression instead shows the highest performance.

# 5 Conclusion

This project aimed to answer: *how accurately can a classifier be learned to maintain anchors of physical objects and which classifier does this best?*. The bar chart shown in Figure 4.4 suggest that there are two accurate anchoring systems with the highest average $F_1$ score obtained by Decision Tree (0.901) for approach 1 and Logistic Regression (0.947) for approach 2.

These scores are in line with those obtained by Persson et al. (2019) who achieved an average $F_1$ score of 0.938 for their best model; Support Vector Machine (SVM). Despite, their being small ob-

served differences between classifiers, our results show no evidence for this difference to be significant between Naive Bayes, Logistic Regression and Decision Tree. A post-hoc test revealed that KNN was significantly worse for both approaches. A similar minor discrepancy can be seen with the results of Persson et al. (2019) who present almost identical performance between a Multi Layer Perceptron (MLP), SVM and KNN (3 neighbours) but slightly lower results for Naive Bayes. KNN's lower performance, is most likely due to $k = 3$ being too low. This could be fixed by more rigours hyperparameter tuning. Generally, it can be concluded that any standard binary classifier will work for bottom-up anchoring, however, to achieve peak performance additional tests are needed.

Comparing approaches, Figure 4.4 clearly illustrates that approach 1 is inferior. The reason for this is that attributes can drastically fluctuate within a very short time frame. This is mainly induced by occlusion. Figure 4.3 shows how the detected contour changes as the hand moves upwards. Since all attributes are extracted from the contour, slight changes, can trigger large differences in certain attributes between time steps. This shortcoming was successfully solved by approach 2 where position is tracked externally and sensitive attributes such as area and direction are not used during those time steps.

It is important to note that the $F_1$ scores, presented when comparing approaches, are averages; Figures 4.5 and 4.9 showed us that approach performance varies with different scenarios. Approach 2 can anchor objects that disappeared as long as their motion follows a linear system of motion, i.e gravity (see Figure 4.8). The Kalman filter brings no advantage when it comes to stationary disappearing objects such as the small ball in the Shell game (see Figure 4.7. This is the primary reason why approach 2 doesn't reach $F_1$ scores above 0.95 for *disappearance*.

When it comes to *stacking* both approaches perform inadequately with $F_1$ scores below 0.90. Approach 2, however, is better as it doesn't anchor stationary objects. When an object is slowing while covered by another object, approach 1 will always fails as the attributes will change too fast. By design approach 2 should avoid these pitfalls. However, in practice we found that the background subtraction is occasionally faulty. For example, shadows from moving object can make a stationary object change in brightness and thus results in a motion detection. This highlights the true difficulty in designing an anchoring system: because it depends on supplementary systems, imperfections may accumulate. To overcome this limitation, a system using semantic relations is required (Persson et al., 2019). For example, if a cup in the Shell game covers a ball, the ball didn't disappear, but rather needs to be bound to the covering cup.

Nonetheless, we were able to build a working anchoring system. Our system is extremely lightweight and easy to train. Unlike the work presented by Persson et al. (2019) our approach does not depend on a CNN or manual labeling of objects for training.

The domains of surveillance and tracking can greatly benefit from anchoring systems (Coradeschi & Saffiotti, 2000). For example, take the task of building a self-driving taxi that receives the following speech command: "follow the blue truck that is now turning right". The goal is not to just follow any car but a blue one which is branded as a truck and is located on the right side of an intersection. Of course a secondary system would be needed to extract the description words using speech analysis. Aside from this, anchoring systems provide a great solution to bridge raw perceptional data, which we humans can not manipulate well, to an abstraction representation that is more applicable to us. A major benefit of anchoring system is the ability of anchors to update regularly. For instance, if a system needs to follow a car which gets muddy from dirt, traditional object recognition systems will fail as they might think the car changed color. The inherent dynamicity of anchoring system can cope with this.

Future work should build on our implementation by incorporating spatial semantic relations into our system. These could be range from hard-coded rules to an advanced inference model. Of course additional attributes can always be investigated. Time, for example, is an attribute of interest that was not examined. Furthermore, the system's efficiency could be improved by deleting old anchors which are no longer relevant and by excluding features that are not necessary for a classification decision.

# References


Bhowmik, T. K. (2015). Naive bayes vs logistic regression: theory, implementation and experimental validation. *Inteligencia Artificial. Revista Iberoamericana de Inteligencia Artificial*, *18*(56), 14–30.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Bredeche, N., Chevaleyre, Y., Zucker, J.-D., Drogoul, A., & Sabah, G. (2003). A meta-learning approach to ground symbols from visual percepts. *Robotics and Autonomous Systems*, *43*(2-3), 149–162.

Coradeschi, S., Loutfi, A., & Wrede, B. (2013). A short review of symbol grounding in robotic and intelligent systems. *KI-Künstliche Intelligenz*, *27*(2), 129–136.

Coradeschi, S., & Saffiotti, A. (2000). Anchoring symbols to sensor data: preliminary report. In *AAAI/IAAI* (pp. 129–135).

Coradeschi, S., & Saffiotti, A. (2003). An introduction to the anchoring problem. *Robotics and autonomous systems*, *43*(2-3), 85–96.

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *The Journal of Machine learning research*, *7*, 1–30.

Elfring, J., van den Dries, S., Van De Molengraft, M., & Steinbuch, M. (2013). Semantic world modeling using probabilistic multiple hypothesis anchoring. *Robotics and Autonomous Systems*, *61*(2), 95–105.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, *82*(Series D), 35–45.

Kotsiantis, S., Kanellopoulos, D., & Pintelas, P. (2006). Handling imbalanced datasets: A review. *GESTS International Transactions on Computer Science and Engineering*, *30*(1), 25–36.

Metropolitan Police Department. (n.d.). *Three-card monte.* DC.gov. Retrieved from `https://mpdc.dc.gov/page/three-card-monte`

Mohanapriya, M., & Lekha, J. (2018). Comparative study between decision tree and knn of data mining classification technique. In *Journal of physics: Conference series* (Vol. 1142, p. 012011).

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., ... Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

Persson, A., Dos Martires, P. Z., De Raedt, L., & Loutfi, A. (2019). Semantic relational object tracking. *IEEE Transactions on Cognitive and Developmental Systems*, *12*(1), 84–97.

Persson, A., Längkvist, M., & Loutfi, A. (2017). Learning actions to improve the perceptual anchoring of objects. *Frontiers in Robotics and AI*, *3*, 76.

Redmon, J., & Farhadi, A. (2018). Yolov3: An incremental improvement. *arXiv*.

Rocha, L., Velho, L., & Carvalho, P. C. P. (2002). Image moments-based structuring and tracking of objects. In *Proceedings. xv brazilian symposium on computer graphics and image processing* (pp. 99–105).

Sadli, R. (2022, Jan). *Object tracking: 2-D object tracking using kalman filter in python.* Retrieved from `https://machinelearningspace.com/2d-object-tracking-using-kalman-filter/`

Saffiotti, A. (1994). Pick-up what? In *Current trends in AI planning.*

Sebastian, P., Voon, Y. V., & Comley, R. (2008). The effect of colour space on tracking robustness. In *2008 3rd ieee conference on industrial electronics and applications* (pp. 2512–2516).

Suzuki, S., & Abe, K. (1985). Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, *30*(1), 32–46.

Zivkovic, Z., & Van Der Heijden, F. (2006). Efficient adaptive density estimation per image pixel for the task of background subtraction. *Pattern recognition letters*, *27*(7), 773–780.

# A Appendix

| | | Approach 1 | | | Approach 2 | | |
|---|---|---|---|---|---|---|---|
| | | *Moving* | *Stacking* | *Disappearance* | *Moving* | *Stacking* | *Disappearance* |
| Bayes | $P$ | 0.944 | 0.749 | 0.789 | 1.00 | 0.826 | 0.978 |
| | $R$ | 0.985 | 0.974 | 0.982 | 0.983 | 0.937 | 0.957 |
| | $F_1$ | 0.964 | 0.839 | 0.871 | 0.991 | 0.867 | 0.947 |
| KNN | $P$ | 0.874 | 0.590 | 0.719 | 0.943 | 0.630 | 0.873 |
| | $R$ | 0.988 | 0.982 | 0.988 | 0.934 | 0.900 | 0.963 |
| | $F_1$ | 0.923 | 0.730 | 0.825 | 0.935 | 0.730 | 0.912 |
| Log Reg | $P$ | 0.937 | 0.750 | 0.754 | 1.00 | 0.831 | 0.950 |
| | $R$ | 0.990 | 0.981 | 0.989 | 0.985 | 0.949 | 0.975 |
| | $F_1$ | 0.962 | 0.841 | 0.850 | 0.992 | 0.878 | 0.960 |
| D-Tree | $P$ | 0.959 | 0.739 | 0.805 | 0.986 | 0.847 | 0.934 |
| | $R$ | 0.990 | 0.984 | 0.988 | 0.959 | 0.949 | 0.946 |
| | $F_1$ | 0.974 | 0.836 | 0.883 | 0.972 | 0.891 | 0.937 |

**Table A.1: Showing average precision ($P$), recall ($R$) and $F_1$ score ($F_1$) rounded to 3 s.f.**