31905 Synthesis in electrotechnology - Jan 19

Dynamical Object Tracking

Student: s181366, Adrià Mompó Alepuz

Supervisors: Silvia Tolu, Marie Claire Capolei

Technical University of Denmark

January 25, 2019

DTU Electrical Engineering
Department of Electrical Engineering

Abstract

In this project a real-time video tracking algorithm for estimating the position of an object in a simulated environment has been implemented. For that, state of the art techniques have been employed, such as Perspective-n-Point pose recovering algorithms and Kalman filter state estimation. The computer vision library OpenCV has been used to process the images and extract the relevant information for the object tracking.

This project has been developed for the department of Electrical Engineering of DTU (Technical University of Denmark), as part of a bigger project aimed to design and test bio-mimetic control architectures under the Human Brain Project scope.

Contents

List of figures

1 INTRODUCTION

Over the recent years, the robotics field has witnessed many improvements in terms of control architectures and algorithms, most of them based on modelling the dynamics of the robot. However, the increasing complexity of the robots being developed and the tasks they are demanded to perform, requires new control techniques that do not fully depend on a model of the robotic system. Instead, they introduce a learning component that allows to bypass part of the dynamics modelling phase, and also give the robot the ability to learn tasks that otherwise would be too time consuming to program. This is where the Neurorobotics field comes into play, which combines the study of neuroscience, artificial intelligence and robotics. With it, exhaustive development driven by different projects is needed in order to make fully functional systems that can compete with the state of the art of robotics control techniques. In one of those projects is where the work in the present project is framed.

1.1 PROBLEM DEFINITION

This project consists of the design of a state observer of a closed-loop control system, using image analysis and object tracking techniques. The system is stated in a simulated environment and focuses on controlling the position of an unstable object actuated by a humanoid robot's arm. Specifically, the robot obtains images with virtual cameras of a ball lying on a flat square surface (table) which is held by its arm, and the control objective is to keep the ball at the center of the surface.

With that, the task to be developed focuses on the implementation of an observer that, from the perspective images taken of the scene, estimates in real time (50 frames per second) the position of the ball relative to the center of the table and by means of tracking algorithms is able to deal with noise in the measurements such as partially occluded images of the objects.

1.2 MOTIVATION

This project is developed in the Neurorobotics subproject of the Human Brain Project (HBP). The control scenario constituted by the simulated robot provides a flexible and convenient way of testing control algorithms that mimic some functions of the human brain, such as the those present in the cerebellum, and which show promising results for complex robotic systems and tasks compared to classical controllers. Some work has already been done in DTU in this research line [1].

With this, the main objective in the present project is to improve the currently in use image analysis algorithms, so an observer that provides a more robust tracking of the ball is obtained. This will provide the student with the opportunity to apply image processing techniques studied in previous courses into a current research problem, as well as to develop and test a state observer, which is a crucial component in the control theory domain, also studied previously. Additionally, the student will also have to research and understand current methods employed in this specific field of object tracking, providing thus a way to broaden the knowledge in this area.

2 STATE OF THE ART

Object tracking consists of two main tasks. First, the detection of an object in a sequence of images taken by a camera by using image analysis algorithms, and second, the estimation of its position (and sometimes other spatiotemporal magnitudes such as speed and acceleration) for all frames, given the direct observations from the images as well as other prior knowledge such as a dynamical model that describes the motion of the object and sometimes its environment.

For the image analysis part, the algorithms used depend highly on the nature of the scene and the object being tracked. In this case, relatively simple techniques such as binary images and contour extraction and analysis will be used to detect the objects on the image.

In this project, the relative positions of the objects in the world must be obtained, so once the objects have been detected in the image, further processing is needed to find their world location. This can be framed as a Perspective-n-Point problem, in which a number of points (n) are defined in the object and detected in its captured image, and the objective is to find the orientation and position of the object with respect to the camera. This will be further explained in the next section.

For the estimation part, also known as filtering, the most successful methods are based on Bayesian filters, which combine the information subject to noise coming from measurements and a transition model (dynamics of the system) to estimate the value of some system states. The two main algorithms that implement this idea are the Kalman filter [2], which assumes the noise sources have normal distributions, and the Particle Filter [3], which can deal with more general non-Gaussian noise process. In this case the Kalman filter is chosen, since the dynamics can be modelled with Gaussian noise sources, as it will be discussed in the implementation section.

2.1 BACKGROUND

2.1.1 ORIENTATION AND POSITION ESTIMATION (PNP PROBLEM)

As stated before, the objective of the PnP problem is to find the orientation and position of an identified object respect to the camera. Specifically, the object is modelled as a rigid body with its own coordinate system in which a set of characteristic points are defined. By detecting these points in the captured images of the object, and knowing the parameters that define the camera (intrinsic camera matrix and distortion coefficients), the rigid body transformation from the object coordinate system (1 in Figure 1) to the camera coordinate system (0 in Figure 1) can be recovered. This transformation is defined by the 3×3 rotation matrix R_1^0 and the 3×1 translation vector t_1^0.

With this, the homogeneous coordinates of a point p^1 originally described in (1) can be expressed in (0) according to the transformation matrix

$$T_1^0 = \begin{bmatrix} R_1^0 & t_1^0 \\ 0_{1\times 3} & 1 \end{bmatrix} \tag{2.1}$$

as follows

$$p^0 = T_1^0\, p^1 \tag{2.2}$$

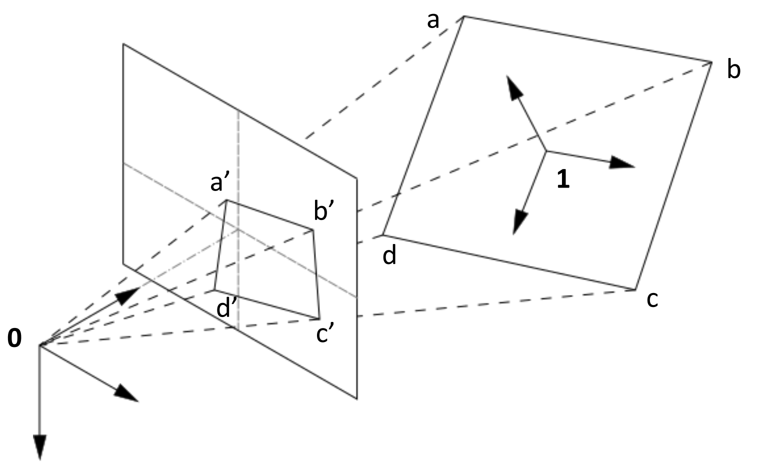


Figure 1. PnP problem definition

If a world point is expressed in the camera coordinate system (0), its image coordinates can be easily obtained by intersecting the vector that defines the location of this point, p^0, with the image plane.

However, if only the location of the point in the image plane is known, its position in the world could be anywhere along the line defined by the vector p^0. This is why, if a known object is to be localized in the world from its image, which is the objective of the PnP problem, multiple points from the object need to be defined; generally, at least 3 are required.

Several methods have been developed over the years to solve this problem. Some use iterative optimization algorithms in which the values of R and t are computed in order to fit the best match between the observed object points in the image and the projection given by the estimated R and t. These are usually solved with Levenberg-Marquardt optimization [7]. Others take an algebraic approach, in which the R and t are found using direct algebraic expressions [4], [5], [6]. In this project, the most recent method is used [6], which efficiently finds the parameters with higher robustness than previous methods.

2.1.2 KALMAN FILTER

Given a discrete time linear state space representation of a system subject to Gaussian noise

$$\begin{aligned} x_{k+1} &= A\,x_k + w_k, \qquad w\sim N(0,Q) \\ y_k &= C\,x_k + v_k, \qquad v\sim N(0,R) \end{aligned} \tag{2.3}$$

where x_k and y_k at the time step k are the true state of the system and the measurement of it respectively, the matrices A, C are known, and v and w are the stochastic signals defined by the known covariance matrices Q and R respectively, then the Kalman filter provides optimal estimates of the state x given their measurements y.

The Kalman filter algorithm performs two steps at every time interval, the filtering and the prediction step. In the filtering step, a prior estimation of the states x_k^- and their covariance matrix P_k^- are given, and with the new information coming from a measurement y_k, it computes the updated estimate of x_k^+ and P_k^+. First, the Kalman gain for the current time interval is found

$$R_{i,k} = CP_k^-C^T + R \tag{2.4}$$

$$K_{fx,k} = P_k^-C^T\left(R_{i,k}\right)^{-1} \tag{2.5}$$

Then, the innovation e_k given by the difference between the new measurement and the state estimate is

$$e_k = y_k - y_k^- = y_k - C\,x_k^- \tag{2.6}$$

and with these, the updated state and covariance are obtained as

$$x_k^+ = x_k^- + K_{fx,k}\,e_k \tag{2.7}$$

$$P_k^+ = P_k^- - K_{fx,k} R_{e,k} \left(K_{fx,k}\right)^T \tag{2.8}$$

In the prediction step, the next time interval state and covariance matrix are computed using the filtered values and the linear model as follows

$$x_{k+1}^- = A\,x_k^+ \tag{2.9}$$

$$P_{k+1}^- = A\,P_k^+\,A + Q \tag{2.10}$$

In many practical applications, since the value of P ends up converging after a given number of iterations, its steady state value is computed, and with it, the steady state Kalman filter gain. This constant P value can be found solving the discrete algebraic Riccati equation

$$P = APA^T + Q - (APC^T)(CPC^T + R)^{-1}(APC^T)^T \tag{2.11}$$

3 METHODS AND MATERIAL

For the development of the project, different tools and resources will be used. They have been divided into hardware and software tools, and data.

Hardware

A personal laptop computer with Windows 10 is used. The microprocessor is an *Intel Core* i7-8750H. This is important to note since it will be the one executing the program, and the performance results will be referred to this microprocessor.

Software

The programming language is Python 3. The different libraries used are

- *OpenCV*, for image processing (the library *cv2* for Python).
- *NumPy* and *SciPy*, for linear algebra functionalities.
- *itertools* and *collections* for different data processing tools.

Data and parameters

The robot simulation is performed in the Neurorobotics Platform. For this project, there has been no access to the platform. Instead, recorded video alongside with some additional data of the different simulation scenarios has been provided to test the algorithms, as well as the virtual camera parameters and the objects dimensions. Specifically,

- Video at 50 fps showing the movement of the ball at different angles of the table and with the camera standing still
- Virtual camera:
    - Resolution: 320x240 px
    - Field of view: 90° horizontally
    - Distortion coefficients: 0 for all of them
- Virtual table: square with side 0.3 m, with elevated borders to keep the ball inside the area under normal operating conditions
- Virtual ball: sphere with radius 0.025 m

4 DYNAMICAL OBJECT TRACKING PROGRAM

In this section the algorithms developed and their implementation in the program will be explained in detail. First, the table and ball detection tasks will be discussed, the result of which is the position measurement of the ball respect to the center of the table. Then, the state space design of the system and the Kalman filter used to estimate the position of the ball according to the measurement and the system dynamics will be presented.

4.1 OBJECT DETECTION

The approach taken to find the position of the ball relative to the center of the table is based assigning a coordinate system to the center of the table (TRF, table reference frame) in which the ball coordinates will be expressed. For that, another common reference will be needed, which in this case is the one that the camera constitutes (CRF, camera reference frame). Specifically, the table is modelled as a rigid body with a coordinate system which position and orientation should be expressed relative to the CRF, while the ball is represented as a point which position in the world, and therefore in the TRF, has to be determined by using the relationship found between CRF and TRF.

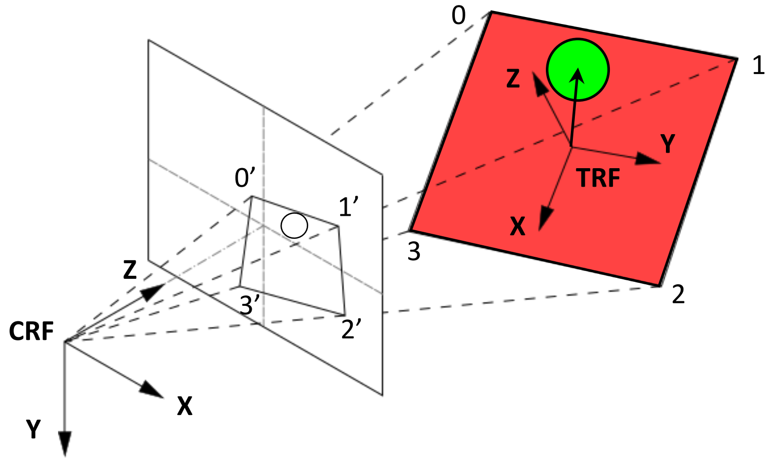


Figure 2. Definition of coordinate frames and points in table

In the following sections, the methods employed to find these values will be explained, presenting both the solution used for the table and for the ball. These are partially interrelated, so first a detailed description for each method separately will be presented, alluding to these relationships at the corresponding steps, and then a diagram summarizing all the operations and their relationships will be provided.

These methods will have as a design requirement to provide to the extent possible reliable outputs under adverse circumstances, such as partially occluded or out of the field of view images of both objects. Although the tracking part will help with this regard, these adverse situations have been considered in the design of the detection of the table and the ball.

4.1.1 TABLE DETECTION

The objective is to solve the PnP problem, in which the rotation matrix R and the translation vector t that define the coordinate transformation from the TRF to the CRF are found. This can be achieved by knowing the real dimensions of the object (table) in the world, its projection onto the camera plane (its image), and the camera parameters. Specifically, n points of the object need to be determined and they must be found in the image. For the camera parameters, the intrinsic camera matrix should be specified, as well as the distortion coefficients.

In the present case, the distortion coefficients are zero, since a virtual camera is being used with no distortion. The intrinsic parameters camera matrix A,

$$A = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{4.1}$$

can be found by knowing that the image size is 320x240 px, the horizontal field of view is 90°, and the focal length is the same both horizontally and vertically. Therefore:

$$c_x = \frac{320}{2} = 160 \tag{4.2}$$

$$c_y = \frac{240}{2} = 120 \tag{4.3}$$

$$f_y = f_x = \frac{320}{2} \cdot \tan\left(\frac{90°}{2}\right) = 160 \tag{4.4}$$

Now, the object points chosen are the four corners of the red table. They are labelled in a clockwise manner. Since the origin of the TRF is located at the center and the axes are those expressed in the Figure 2, the points $p_i = \{p_{i,x} \;\; p_{i,y} \;\; p_{i,z}\}$, $i = 0,1,2,3,$ are assigned the following coordinates

$$P_{obj} = \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \begin{bmatrix} p_{0,x} & p_{0,y} & p_{0,z} \\ p_{1,x} & p_{1,y} & p_{1,z} \\ p_{2,x} & p_{2,y} & p_{2,z} \\ p_{3,x} & p_{3,y} & p_{3,z} \end{bmatrix} = \begin{bmatrix} -L/2 & -L/2 & 0 \\ -L/2 & L/2 & 0 \\ L/2 & L/2 & 0 \\ L/2 & -L/2 & 0 \end{bmatrix} \tag{4.5}$$

where L is the length of the side of the table. The value assigned to L will determine the distance t between CRF and TRF obtained from solving the PnP. This is however not relevant for the objective of this project, in which only the rotation matrix R will be used, so the an arbitrary positive number can be assigned to L.

Finally, these 4 object points must be identified in the image. The proposed solution is based on finding the 4 segments that define the borders of the table and then obtaining the intersection of these that correspond to the vertices of the table. This corresponds to the image analysis part of the program, and the sequence of operations implemented to achieve this is presented next.

4.1.1.1 Implementation

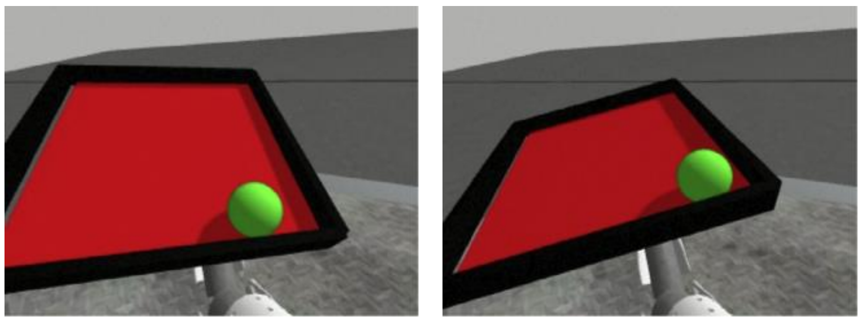


Figure 3. Pair of images used to illustrate the table detection

The first operation performed after an image is loaded is a blurring using a Gaussian filter. This is needed in order to remove high frequency noise from the image, which can produce undesired results when processing the image with the further algorithms.

With this, in order to retain solely the shape corresponding to the table while removing all other information from the image, a binary image representing the red area of the table is obtained. Since no other object in the field of view presents similar colors, the easiest way to obtain the binary image is by defining the color range that the board can take, considering all situations in which the intensity changes due to shadows, and then performing an operation that assigns the white value to the pixels in such range and black to the rest. The best way to define this range is by first changing the color format of the image from RGB to HSL (hue, saturation, lightness) or HSV (hue, saturation, value). These formats allow to easily select the hue (in this case red), and then the variations of saturation and lightness or value due to shadows can be adjusted according to the observed images. In this case the HSL format is selected.

In OpenCV, the function *cvtColor* with the code *COLOR_BGR2HLS* performs the transformation from RGB format to HSL. Then, the desired range is defined as two NumPy arrays containing the lower and upper values of each channel of the image (hue, saturation, lightness), and with this, the function *inRange* from OpenCV extracts the binary image which white pixels correspond to such range. An additional set of operations often performed on binary images to remove noise are the morphological transformations. In this case the transformations of opening and closing have been implemented, with the OpenCV function *morphologyEx* and the tags *MORPH_OPEN* and *MORPH_CLOSE*.

Now, once the region of interest corresponding to the table has been identified, a sequence of operations to detect the segments that define the borders is required. A first approach considered using line detectors on the binary image found, among them the Hough line transform [8] and the Line Segment Detector (LSD) [9] were tested. The Hough transform proved to be difficult to tune and was very sensitive to noise that made the borders not to be completely straight. Although the LSD provided reasonably good results and needed fewer tuning parameters, it still showed some sensitiveness to noise and required additional processing when the borders were segmented in two parts due to the presence of the ball.

Because of this, the final implemented approach relies on extracting the contours of the binary image and processing them to find the relevant lines that compose the borders. The main idea behind this is to obtain the convex shape that approximates the contour found, represented with as few points as possible, in order to capture the most significant vertices of the shape, that will correspond in the general case to the vertices of the table.

This can be achieved using three OpenCV functions. The first one, *findContours*, with the arguments *RETR_EXTERNAL* and *CHAIN_APPROX_SIMPLE*, returns the external contour of the binary image with the set of points that capture every straight line (this is, every section of the contour corresponding to a straight segment will be represented just by the start and end points of that segment). Since the process of obtaining the binary image leads to a shape with rough borders, the set of straight lines that compose such shape will actually be large, and so will be the set points representing the contour. Besides, it will also capture the concave region corresponding to the presence of the ball in a border of the table.

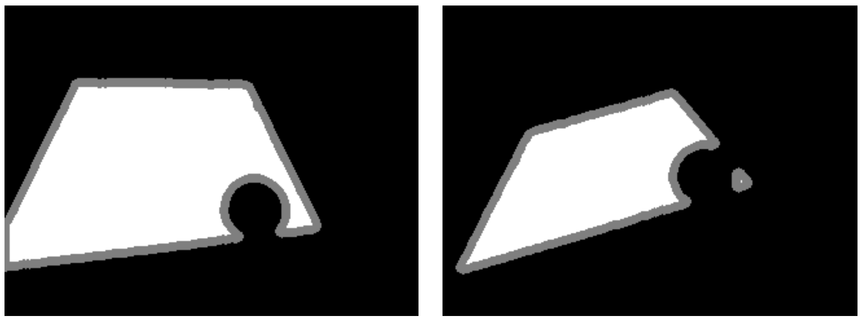


Figure 4. Found contours of table, represented in grey over the binary image

This all can be addressed with the two remaining functions. The first, *convexHull*, takes the previously obtained contours and returns the set of points defining the contour with no concave regions. The second, *approxPolyDP*, approximates the convex contour with a given precision *epsilon*, in this case set to 1% of the width of the image, which represents the maximum distance in pixels from the original to the processed contour. This effectively removes the roughness of the borders caused by noise, providing a set of points that captures the relevant corners of the shape.

It must be noted that in some situations more than one contour may be detected, so to deal with this situation the contour with largest area is selected.

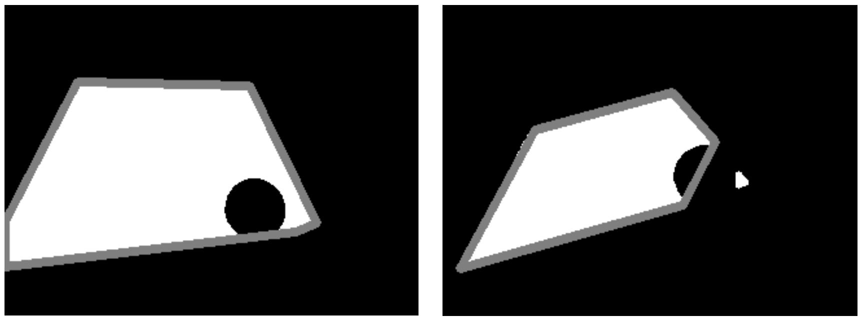


Figure 5. Processed contours. 6 segments identified in the left image and 5 in the right one.

After all these operations, there still exist some situations in which the number of points obtained will still be greater than 4. These are the cases when the ball is at one corner of the table, or when part of the corner of table is out of the field of view. The way to suppress these segments, comprises two main operations. The first one requires previous image processing of the ball region, which will be explained in the next section. It takes the center of the ball and the measured radius in the image (in pixels), and checks which points of the contour are close to the ball. If they meet a certain proximity criterion, they are discarded as candidates to form a segment representing the border of the table. The second operation considers all possible segments that can be formed out of every pair of consecutive points (except from the one that has been excluded due to the presence of the ball) and takes the four that have the greatest length.

All these operations effectively select the four relevant segments that comprise the borders of the table. The last remaining step is to find the four intersections that correspond to the four vertices of the table. Since given 4 oblique segments the number of possible intersections between them is 6, there will be 2 that will occur far from the area corresponding to the table, and 4 (the relevant ones) that will be closer.

Knowing this, an algorithm to find the intersections and their proximity to the segments is developed. The intersections are obtained by following the theory developed for the case of two lines defined by two points each [10]. Additionally, the Bezier parameters for linear curves are computed to assess the proximity to the intersection point and to help finding the intersection [11]. The algorithm is as follows:

- Given two lines a and b, they are defined by the points $a_1 = (a_{1,x}, a_{1,y})$ and $a_2 = (a_{2,x}, a_{2,y})$ for the first line and $b_1 = (b_{1,x}, b_{1,y})$ and $b_2 = (b_{2,x}, b_{2,y})$ for the second line. The vectors representing these lines are obtained as $v_a = a_1 - a_2,\ v_b = b_1 - b_2$.
- Now, the vectorial product between these two vectors is obtained, and if the result is lower than a certain threshold, the lines are considered parallel and they don't intersect. If it is greater, the Bezier parameters for each line are obtained.
- For the Bezier parameters, the vector $ab = a_1 - b_1$ is computed first. Then the parameters are defined as
$$t_a = ab \times v_a$$
$$t_b = ab \times v_b$$
- Finally, the intersection point can be obtained as $p = a_1 - t_b v_a$

If the parameters t_a, t_b are close to 0 or 1, it implies that the intersection will be close to the endpoints of the segments. Additionally, for the set of operations defined above, if the intersection is actually close to the segments, in all cases one of these two parameters will take a value close to 1 and the other close to 0. However, if the intersection is far from the segments, one parameter will take values much lower than 0 and the other much greater than 1. With this, the condition established to discern the points that are part of the vertices of the square from those that aren't, is to assess the product $|t_a| \cdot |t_b|$. If it is lower than a threshold, in this case set to 1.5, then the intersection is considered a vertex of the table, and the opposite otherwise.

The last step once the four vertices have been found is to sort them to match the distribution specified above (see Figure 2). In this case, since the table doesn't rotate significantly with respect to the camera, the four points always keep a certain relative position with respect to each other. Specifically, the two leftmost points always have lower X (horizontal) coordinate values than the two rightmost points. In turn, for each of these two pairs of points (left and right), the points never change their vertical positions (Y coordinate). Because of this, all points can be identified by first sorting them horizontally and selecting the first two and the last two, and then sort these pairs again vertically. This, as stated above, will solve the identification problem only when the table doesn't rotate. If this wasn't the case, an active tracking of the points would be needed, in which an algorithm would search the points close to the position they were found in the previous frame, and according to some motion model.

Finally, with all four points having been identified in the image, the PnP problem can be solved. In OpenCV, the function *solvePnP* from OpenCV implements several of the existing methods to solve this problem. As stated before, the method chosen here is the one proposed by Tong Ke and Stergios Roumeliotis [6], which is specified in the *solvePnP* function by passing the *SOLVEPNP_AP3P* argument.

With this, the function can be called with the four identified vertices in the image, the 3D object points, the intrinsic camera matrix and the distortion coefficients. This function returns the vector r expressing the rotation in the axis-angle format and the translation vector t are found. To represent the rotation as a rotation matrix R, the OpenCV function *Rodrigues* is used, which takes r and outputs R.

As it can be observed in the images below, the four vertices are identified under different perspective conditions, and the coordinate system location given by R and t is correctly found. The red line represents the X axis, the green one the Y axis and the blue one the Z axis.

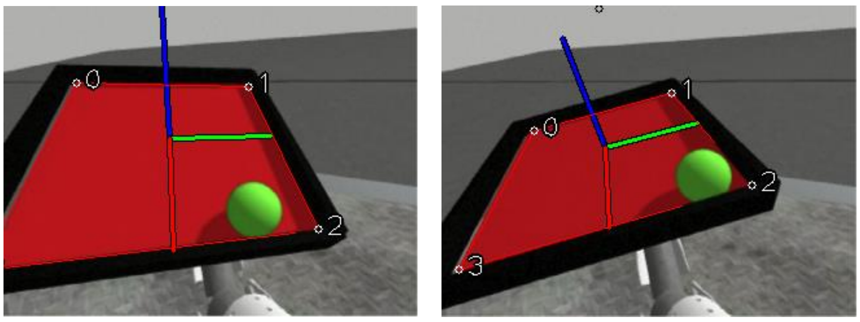


Figure 6. Final table detection result.

4.1.2 BALL DETECTION

As stated previously, the objective is to find the position coordinates of the ball relative to the center of the table, this is, expressed in the table coordinate system (TRF). This is a geometry problem which can be solved mainly in two ways. The first assumes the radius of the ball is known and it localizes its center in the world, and therefore in the TRF. The second takes the radius as an unknown parameter, but it imposes as a constraint that the ball should be on the table, and localizes the ball on it as well as it estimates the radius. The first approach would be the preferable method if the radius was known, but here the more general approach of an unknown radius is chosen.

The key idea behind this approach lies in finding the intersection between the ray v that passes through the optical center and the ball center, and the plane which is parallel to the table and contains the center of the ball (see Figure 7).

The ray v can be defined just from the information present on the image. Knowing that the perspective projection of the ball onto the image plane corresponds to an ellipse (or a circle if the projection is aligned with the principal point), the vectorial sum of the unit vectors v_a and v_b that connect the optical center with the vertices of the ellipse, will define the direction of the ray v (see Figure 8).

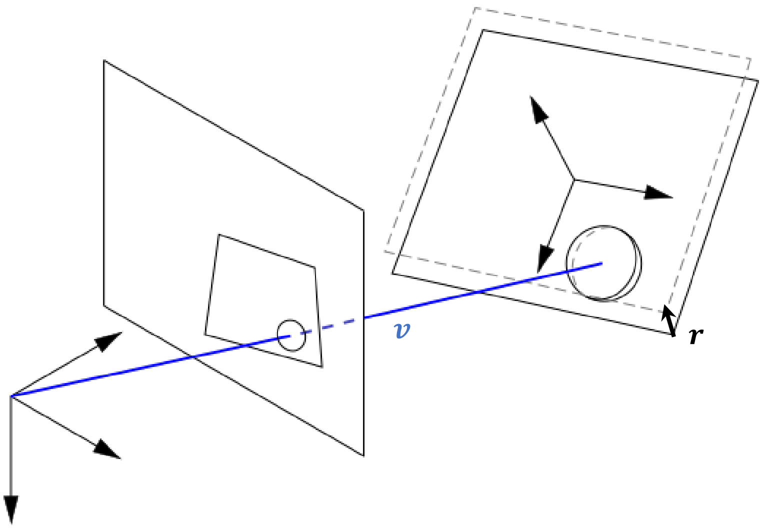


Figure 7. Ray v connecting the optical center with the ball center

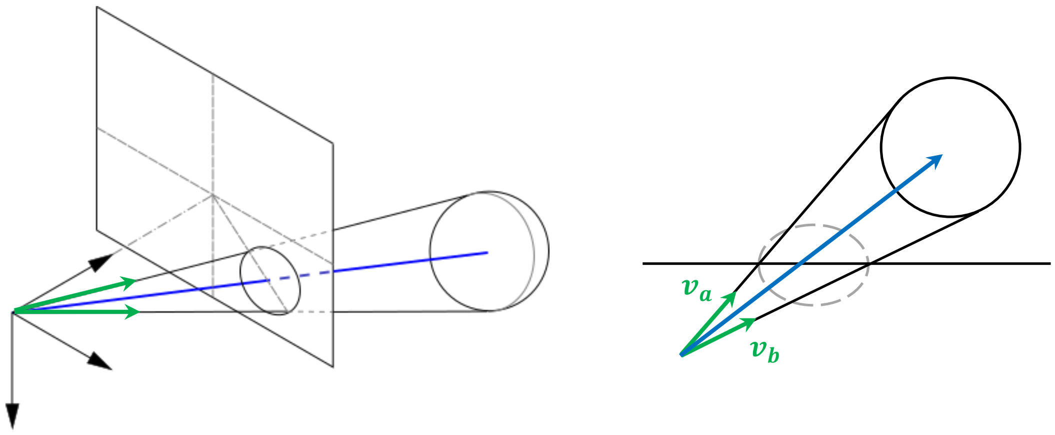


Figure 8. Vectors v_a and v_b defined by the vertices of the ellipse

Now, given a point p_{CRF} in the coordinate system CRF (which may be represented as a vector going from the origin of CRF to such point), it can be expressed in the other coordinate system TRF as:

$$p_{TRF} = \left(R_{TRF}^{CRF}\right)^T \left(-t_{TRF}^{CRF} + p_{CRF}\right) \tag{4.6}$$

For the sake of simplicity, the system CRF will be referred as 0, and TRF as 1. The rotation matrix and the translation vector will be written as R and t respectively.

$$p_1 = R^T(-t + p_0) \tag{4.7}$$

With this, the vector that goes from the optical center to the ball center can be defined as $p_0 = d\ \hat{v}_0$, with d being the absolute distance and $\hat{v}_0$ the unit vector of the ray v previously obtained. This vector may be expressed in the TRF as

$$\begin{aligned} p_1 &= -R^T t + R^T d\ \hat{v}_0 \\ &= t' + d\ \hat{v}_1 \end{aligned} \tag{4.8}$$

with t' being the origin of the CRF expressed in the TRF, and $\hat{v}_1$ the unit vector of the ray expressed in TRF. If the z component of this expression is taken and the constraint of the ball lying on the table is applied, it results in

$$p_{1,z} = r = t'_z + d\,\hat{v}_{1,z} \tag{4.9}$$

This equation has two unknowns, d and r, so another equation relating these variables is needed. It can be obtained by analysing the geometrical relationship between the radius of the ball and the distance.

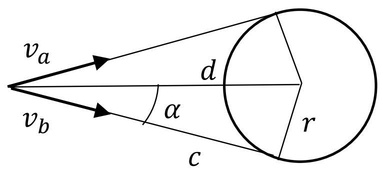


Figure 9. Distance to center of the ball and radius relationship

From the figure above it is known that

$$d^2 = r^2 + c^2 \tag{4.10}$$

The distance c can be obtained as $c = \frac{r}{\tan(\alpha)}$, and the angle is computed from the dot product between v_1 and v_2

$$\alpha = \frac{1}{2}\arccos\left(\frac{v_1 \cdot v_2}{|v_1| \cdot |v_2|}\right) \tag{4.11}$$

Taking this into account, the equation can be rewritten as

$$d = r\sqrt{1 + \frac{1}{\tan(\alpha)}} \tag{4.12}$$

With this it is possible to solve for the distance and the radius with the two equations defined. The final step to obtain the coordinates of the ball in the TRF is to feed the d found into the 4.8. By taking the two first entries of the resulting vector (components x and y), the position of the ball relative to the center of the table is found.

Next, the image analysis part of the program regarding the detection of the ball is explained. The objective is to find the two vertices of the ellipse in order to build the v_1 and v_2 vectors, as well as to extract its center in the image and the major axis length to assist in the detection of the table borders, as mentioned before.

4.1.2.1 Implementation

The program implementation for the detection of the ball starts with the same smoothened image stated in the section of the table image analysis. The next step is to extract the region of the image corresponding to the ball as a binary image. Since the ball is the only green object in the field of view, this can be achieved in the same way that was described for the case of the table, this is, by defining the color range of the ball and obtaining the corresponding binary image with the function *inRange*.

With the binary image representing the area of the ball as an ellipse, its center and semi-major axis length can be found as follows. First, its contour is obtained with the OpenCV function *findContours*. Next, these contours are fed to the function *minEnclosingCircle*, which returns the center coordinates and the radius of the minimum area circle that encloses the contour. Since the contour is an ellipse, the radius of the enclosing circle corresponds to the length of the semi-major axis, and the center is the same for both.

These two parameters (center and radius of the enclosing circle) are used by the algorithm of the table which selects the segments corresponding to the borders, as stated before.

Now, in order to obtain the location of the vertices of the ellipse in the image, the following operations are implemented:

- Find the unit vector that points from the center of the image to the center of the enclosing circle. If the center of the enclosing circle is at the center of the image, the unit vector can take any direction. In this case the horizontal direction is chosen.
- Add and subtract this vector times the radius of the enclosing circle to the center of it, so two points are obtained, corresponding each one to one vertex of the enclosing circle.

With this, the vectors v_1 and v_2 are built, and after the rotation matrix R and the translation vector t are obtained from the image analysis of the table, the previously explained procedure to obtain the location of the ball respect to the center of the table is implemented.

4.1.3 Image analysis summary

The following diagram summarizes all the operations explained comprising the image analysis part of the program.

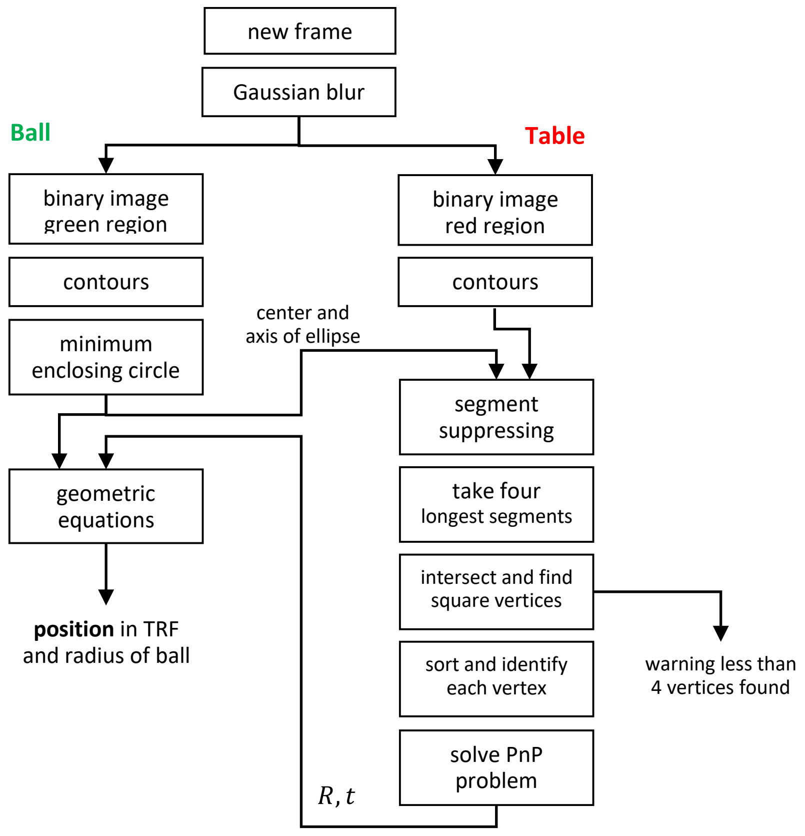

new frame
Gaussian blur
Ball
Table
binary image green region
contours
minimum enclosing circle
geometric equations
position in TRF and radius of ball
center and axis of ellipse
binary image red region
contours
segment suppressing
take four longest segments
intersect and find square vertices
warning less than 4 vertices found
sort and identify each vertex
solve PnP problem
R, t

Figure 10. Summary of the image analysis (object detection) part of the program

4.2 Object tracking

As stated previously, the algorithm chosen to perform the tracking of the ball is the Kalman Filter. It will provide a way to optimally estimate the position of the ball according to a model of the dynamics of the system and the noise characterization of the measurements and the states of the system.

In the present case, the state is first selected as the set of variables which dynamics can be modelled with the available information, and after that an extended model which includes disturbances as states to be estimated by the Kalman Filter will be implemented. The model will be defined in a continuous time state space form, and then discretization will be used to implement it in the Kalman filter.

4.2.1 State space and Kalman filter modelling

The dynamics of a solid ball rolling down an inclined surface under the effect of gravity can be summarized with the expression

$$a = \frac{5}{7} g \sin(\theta) \tag{4.13}$$

where a is the downhill acceleration of the ball in the direction parallel to the surface, g is the gravitational acceleration on Earth, and θ is the angle of the surface with respect to the ground.

Apart from the gravity, the ball will experience additional forces coming from the movement of the table that will affect its dynamics. However, these are more complex to model and some of them are not possible to determine if the movement of the table is not known accurately. In this case, no information is given about the control actions performed on the table, so it becomes difficult and error-prone to estimate the dynamics of the table just from the observations of its position.

Because of this, the initial set of state variables are chosen to be the position p and velocities v in both coordinates X and Y of the table plane, defined by the following vector

$$x = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} p_x \\ v_x \\ p_y \\ v_y \end{bmatrix} \tag{4.14}$$

which satisfies the following dynamics equations

$$\dot{x} = \begin{bmatrix} \dot{p}_x \\ \dot{v}_x \\ \dot{p}_y \\ \dot{v}_y \end{bmatrix} = \begin{bmatrix} v_x \\ a_x \\ v_y \\ a_y \end{bmatrix}, \quad \begin{aligned} a_x(t) &= \frac{5}{7} g \sin\big(\theta_x(t)\big) \\ a_y(t) &= \frac{5}{7} g \sin\left(\theta_y(t)\right) \end{aligned} \tag{4.15}$$

The angles θ_x and θ_y are a function of time, and correspond to the inclination of the table's X and Y coordinate axes with respect to the ground, respectively. These are taken from the measurements of the table rotation respect to the camera, given by $R = R_{table}^{cam}$, and assuming the rotation matrix R_{cam}^{gnd} from the camera to the ground is also known. At a given time instant, it is possible to obtain directly the expressions $\sin(\theta_x)$ and $\sin(\theta_y)$ by taking the X and Y components respectively of the Z vector of the ground coordinate frame expressed in the table coordinate frame. This is

$$\begin{bmatrix} \sin(\theta_x) \\ \sin(\theta_y) \end{bmatrix} = \begin{bmatrix} \left(Z_{gnd,table}\right)_x \\ \left(Z_{gnd,table}\right)_y \end{bmatrix} \tag{4.16}$$

$$\begin{aligned} Z_{gnd,table} &= \left(R_{cam}^{gnd} R_{table}^{cam}\right)^T Z_{gnd} \\ Z_{gnd} &= [0 \quad 0 \quad 1]^T \end{aligned} \tag{4.17}$$

Considering that the measured variables y_1, y_2 are the X and Y positions of the ball in the TRF,

$$y = \begin{bmatrix} y_1 \\ y_2 \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \end{bmatrix} \tag{4.18}$$

the following state space representation of the system can be built

$$\begin{aligned} \begin{bmatrix} \dot{p}_x \\ \dot{v}_x \\ \dot{p}_y \\ \dot{v}_y \end{bmatrix} &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ v_x \\ p_y \\ v_y \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{5}{7}\sin\left(\theta_x(t)\right) \\ 0 \\ \frac{5}{7}\sin\left(\theta_y(t)\right) \end{bmatrix} g \\ \begin{bmatrix} p_x \\ p_y \end{bmatrix} &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ v_x \\ p_y \\ v_y \end{bmatrix} \end{aligned} \tag{4.19}$$

Establishing

$$\begin{aligned} A &= \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix}, \qquad B(t) = \begin{bmatrix} 0 \\ \frac{5}{7}\sin\left(\theta_x(t)\right) \\ 0 \\ \frac{5}{7}\sin\left(\theta_y(t)\right) \end{bmatrix}, \qquad u = g \\ C &= \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \end{aligned} \tag{4.20}$$

the system can be written as

$$\begin{aligned} \dot{x}(t) &= A\, x(t) + B(t)\, u \\ y(t) &= C\, x(t) \end{aligned} \tag{4.21}$$

This corresponds to a linear time variant (LTV) system, since the matrix B must be obtained every time new information of the table rotation is available (every frame in the discrete case).

Now, the presence of unknown control actions is modelled as a stochastic disturbance that affects the speed of the ball. Two components of such disturbance are modelled, a high frequency one, $w_1(t)$, corresponding to white noise with standard deviation σ_w, and another with slower dynamics, $w_2(t)$, corresponding to a filtered white noise with the same standard deviation and a time constant β. Their autocorrelation functions can be written as

$$R_{w_1 w_1} = \sigma_w^2\, \delta(\tau) \tag{4.22}$$

$$R_{w_2 w_2} = \sigma_w^2\, e^{-\beta|\tau|} \tag{4.23}$$

where τ is the time interval between two sampling instants of the signal, and δ is the Dirac delta function. These signals can be expressed as a function of a white noise source $\varepsilon(t) \sim \mathrm{WN}\left(0, \delta(t)\right)$, with mean 0 and intensity 1, as follows

$$w_1(t) = \sigma_w \, \varepsilon(t) \tag{4.24}$$
$$w_2(t) = -\beta \, w_2(t) + \sqrt{2\beta}\sigma_w \, \varepsilon(t) \tag{4.25}$$

These will affect the velocity by adding them to its differential equation

$$\dot{v}(t) = a(t) + w_1(t) + w_2(t) \tag{4.26}$$

where $a(t)$ is the acceleration previously derived of the ball rolling on a ramp.

All of this can be represented in an extended state space system which includes a white noise input to the system and the new variables $w_{2,x}$, $w_{2,y}$ (which will be referred to as w_x and w_y for simplicity), which dynamics are included in the extended state transition matrix A_e.

$$\begin{bmatrix} \dot{p}_x \\ \dot{v}_x \\ \dot{p}_y \\ \dot{v}_y \\ \dot{w}_x \\ \dot{w}_y \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -\beta & 0 \\ 0 & 0 & 0 & 0 & 0 & -\beta \end{bmatrix} \begin{bmatrix} p_x \\ v_x \\ p_y \\ v_y \\ w_x \\ w_y \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{5}{7}\sin(\theta_x(t)) \\ 0 \\ \frac{5}{7}\sin\left(\theta_y(t)\right) \\ 0 \\ 0 \end{bmatrix} g + \begin{bmatrix} 0 & 0 \\ \sigma_w & 0 \\ 0 & 0 \\ 0 & \sigma_w \\ \sqrt{2\beta}\sigma_w & 0 \\ 0 & \sqrt{2\beta}\sigma_w \end{bmatrix} \varepsilon(t)$$
$$\begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ v_x \\ p_y \\ v_y \\ w_x \\ w_y \end{bmatrix} \tag{4.27}$$

The noise source $\varepsilon(t)$ is modelled as a column vector of two uncorrelated white noise signals, each of them affecting a different coordinate (X and Y) of the table.

The extended LTV system is then

$$\begin{aligned} \dot{x}_e(t) &= A_e \, x_e(t) + B_e(t) \, u + B_\varepsilon \, \varepsilon(t) \\ y(t) &= C_e \, x_e(t) \end{aligned} \tag{4.28}$$

Finally, a noise source in the measurement is added. It is modelled as two uncorrelated white noise signals with zero mean and intensity Λ, also defined by the standard deviation $\sigma_v = \sqrt{\Lambda}$

$$\lambda(t) = \mathrm{WN}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} \Lambda \, \delta(t) & 0 \\ 0 & \Lambda \, \delta(t) \end{bmatrix}\right) \tag{4.29}$$

With this, the complete extended LTV model considering all noise sources is

$$\begin{aligned} \dot{x}_e(t) &= A_e \, x_e(t) + B_e(t) \, u + B_\varepsilon \, \varepsilon(t) \\ y(t) &= C_e \, x_e(t) + \lambda(t) \end{aligned} \tag{4.30}$$

After the continuous time LTV model is defined, a discrete time version should be obtained in order to implement the Kalman filter in the program. The discrete time matrices $\bar{A}_e$, $\bar{B}_e$, $\bar{C}_e = C_e$, $\bar{B}_\varepsilon = I_{6\times 6}$ representing the LTV discrete system are obtained, as well as the discrete noise covariance matrices $\bar{Q}$ and $\bar{R}$ for the process and measurement noise respectively.

The last step is to obtain the Kalman filter gain matrix based on these discrete state space and noise matrices. The stationary case is assumed here, so following that explained in the background section, the discrete algebraic Riccati equation is solved using the *solve_discrete_are* function from the linear algebra module of the library *scipy* to obtain the stationary state covariance matrix P, and with it the Kalman filter gain K_{fx} is found.

$$R_i = C_e P C_e^T + \bar{R} \tag{4.31}$$
$$K_{fx} = P C_e^T R_i^{-1} \tag{4.32}$$

4.2.1.1 Kalman Filter implementation

For the implementation of the Kalman filter in the program, a custom class is built, which stores the information regarding the state space, the Kalman filter gain, and the filtered and predicted states from last time step. It also has two main functions, *filter* and *predict*, which implement the equations described in the background section for the filtering and prediction steps respectively. The first one takes a new measurement of the ball position and outputs the filtered state, and the second one takes a new input (which in this case is always the same, the gravity acceleration), and outputs the predicted state for the next time step.

In the prediction step, an operation that computes the next N output predictions according to the dynamics of the system and the last filtered state value is added. This becomes useful when assessing the quality of the modelled dynamics and could be used in a model predictive control setup, in which a controller optimizes the next N control actions.

According to the physical properties of the system, the ball position cannot exceed the boundaries of the table under normal working conditions. Additionally, the collision with the borders of the table is inelastic, meaning that all kinetic energy is lost. Because of this, in the prediction step, another operation is added to account for these restrictions. When the predicted position value of the ball center exceeds the physical limit (expressed in the TRF, half of the table side minus the ball radius), the position is set to such value, and the corresponding velocity is set to zero. This effectively solves undesired predicted and filtered values that surpass such physical constraints.

The values given to the standard deviations of the process noise σ_w and the measurement noise σ_v will determine how the filtering step in the Kalman filter weights the information from the model dynamics against the measurements. If $\sigma_w \gg \sigma_v$, the uncertainty of the model is much higher so the Kalman gain K_{fx} will be greater, and the filtered state will be close to the measured one. Conversely, if $\sigma_v \gg \sigma_w$, the measurements are less trustworthy than the model itself, so K_{fx} will be lower and in the filtering step the state values will not change much.

As additional implementation remarks:

- It should be noted that the discrete input matrix $\bar{B}_e$ must be computed in every iteration before the prediction step, after the new rotation measurement of the table has been obtained.
- In the case that the ball is not properly detected for a given number of time steps, the Kalman filter returns the predicted position values for such time steps.

4.2.2 KALMAN FILTER SUMMARY

The integration of the Kalman filter with the program is showed in the following diagram

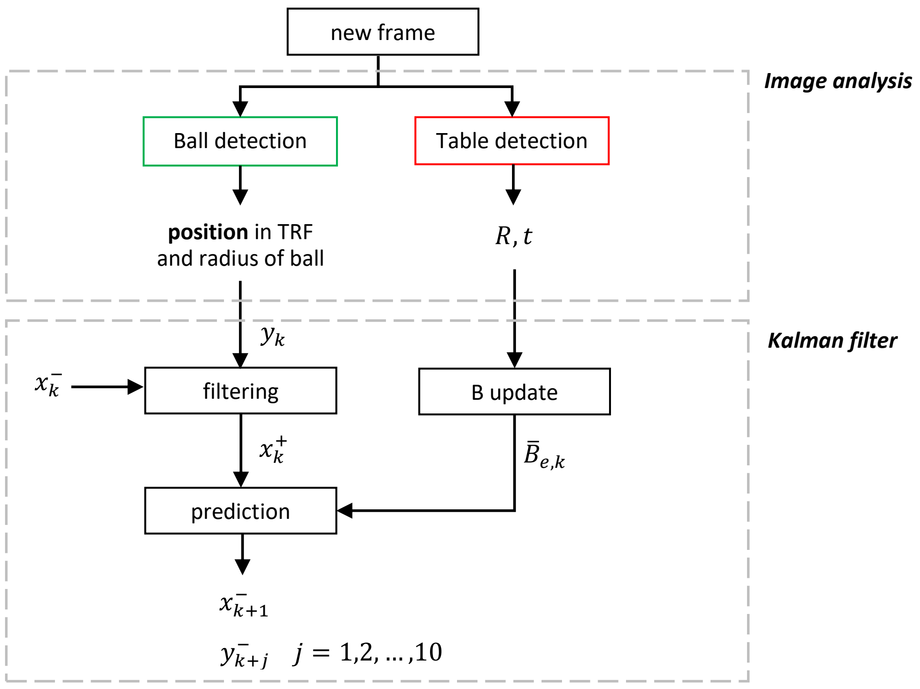


Figure 11. Summary of whole program, including Image analysis and Kalman filter

5 Results and analysis

The data provided for testing the accuracy of the ball detection and tracking algorithms did not include the ground truth of its position. This is why, for assessing the quality of the program outputs, situations in which the true position of the ball can be deduced visually have been used to check the accuracy of the computed position. Additionally, the filtered position values have been compared against the direct measurements to assess the effectiveness of the Kalman filter, and different σ_w to σ_v ratios have been tested to appreciate its outcome on the tracking task and to obtain the optimal ratio for this project (it is defined as optimal the ratio that makes the output resemble more the true visually observed trajectory). Other visual markers have been implemented as well to provide additional means of analysis.

5.1 Results

Next, a set of figures corresponding to three consecutive instants of a specific trajectory described by the ball is presented. Two cases are shown, the first corresponding to the optimal ratio found for σ_w and σ_v, in which $\sigma_w \gg \sigma_v$, and the second for $\sigma_w = \sigma_v$.

The figures on the left side represent a top view of the trajectories of the ball on the table for the last 80 sampling intervals. The red trajectory corresponds to the raw measured position extracted from the detection algorithm, while the blue one shows the filtered value obtained with the Kalman filter. The evolution of the trajectory is given by the color gradient of the lines plotted; for the measured position, the color evolves from magenta (oldest) to red (most recent), and for the filtered position, the color goes from cyan to blue. The black squares represent the borders of the table (thick lines) and the physical limit that the center of the ball can reach (thin lines), which is separated from the table border a distance equal to the ball radius.

On the right side, the figures show the captured images with additional processed information added on top. For the table, the four identified vertices and the measured position and orientation of the coordinate system (X axis in red, Y axis in green, Z axis in blue) are represented. For the ball, a green bounding rectangle is added, as well as a cyan segment representing the radius of the ball in the world going from its center to the contact point with the table, and a set of 10 circles representing the future position predictions of the ball center given by the Kalman filter, going from cyan (next prediction) to blue (ten steps head prediction).

- **Optimal case, $\sigma_w = 1, \sigma_v = 0.005$**

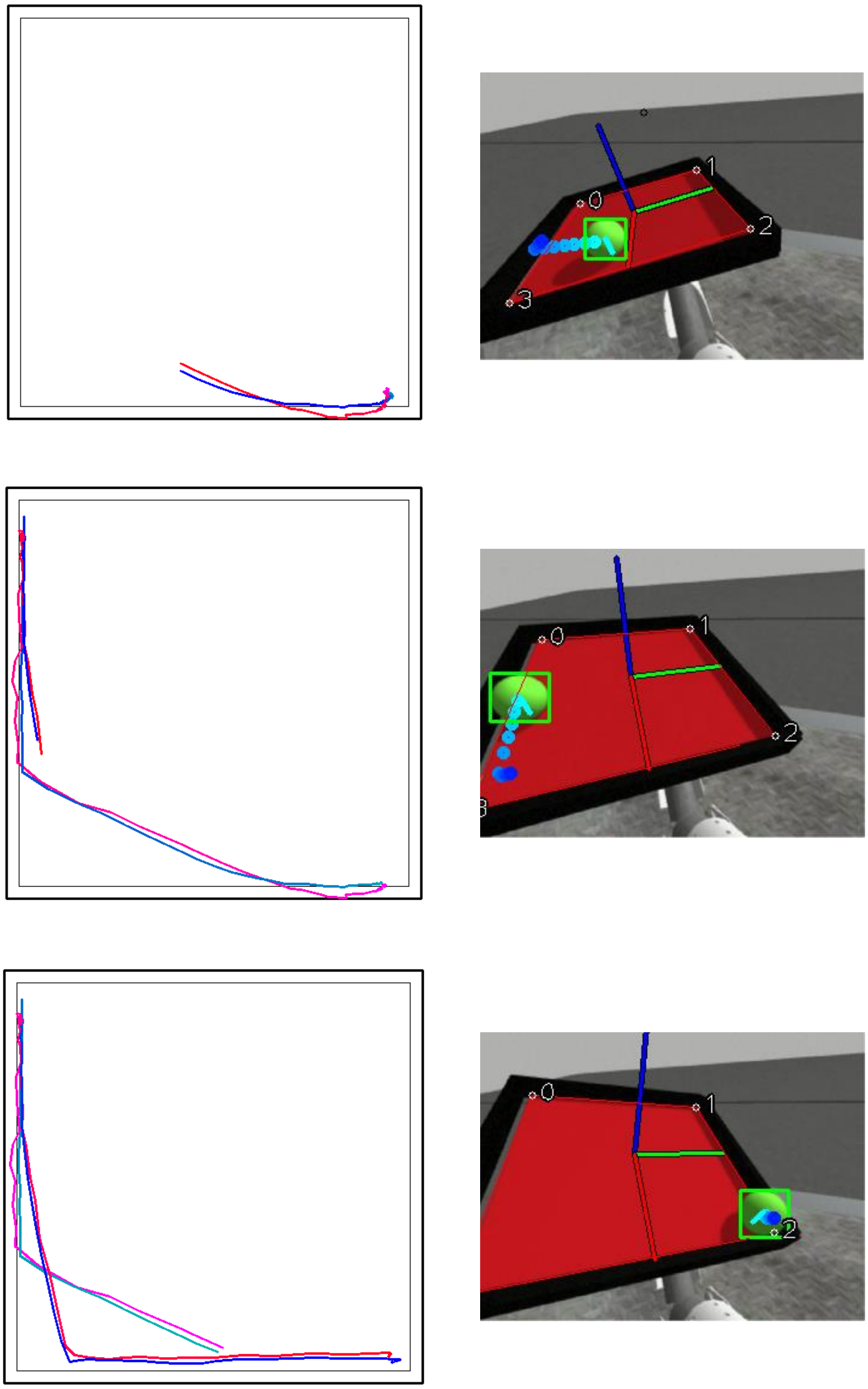
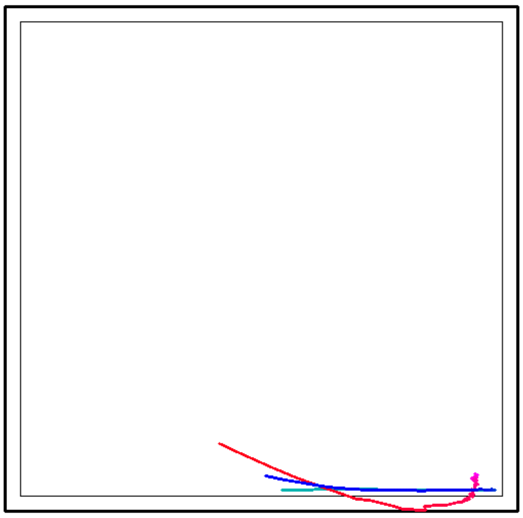
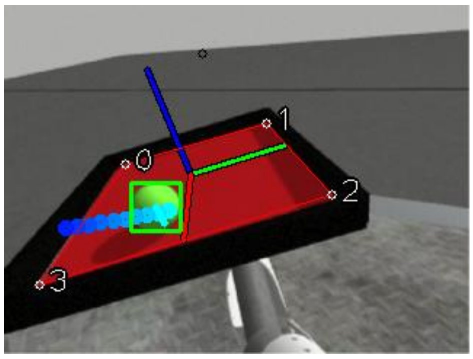
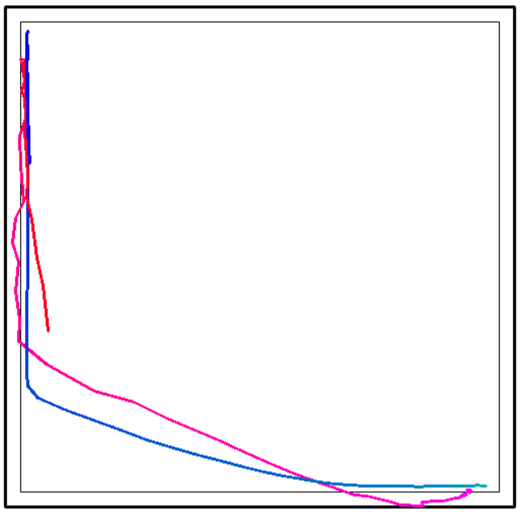


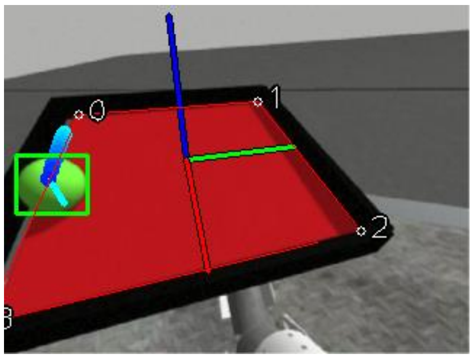
Figure 12. Trajectory for the optimal case

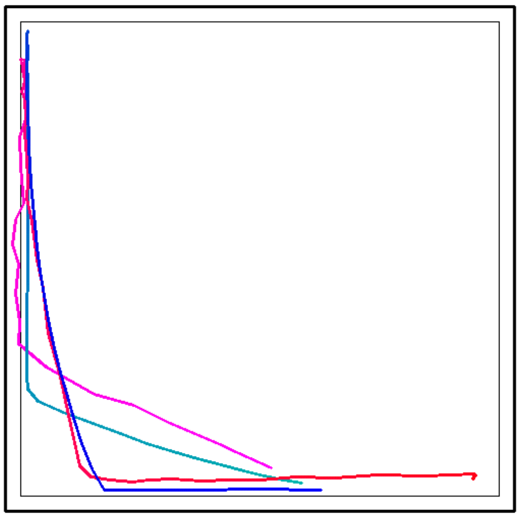
- **Non-optimal case, $\sigma_w = \sigma_v = 1$**











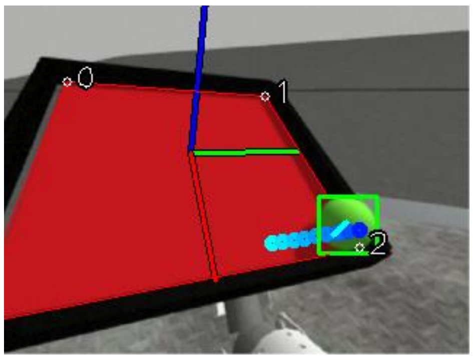


Figure 13. Trajectory for the non-optimal case

5.2 Discussion

As it can be observed, the filtered values clearly improve the quality of the estimated ball position for the optimal case.

- Firstly, it effectively removes high frequency noise, resulting in a smoother trajectory estimation which resembles more the true motion of the ball.
- Secondly, by taking into account the border constraints, the filtered values do not surpass the physical limits that the ball can reach due to the collision with the table border.
- In the third place, the modelled dynamics help estimating the position of the ball in some cases in which the observations direct measurements differ from the true position. This is the case of the top left corner of the square, which the ball reached during the simulation but the measurements (as shown in red) fail to locate its position properly, since they show that the ball didn't reach the top border. The filtered values are closer to this border, and therefore to the true position.
- Lastly, from the left pictures, it can be seen that the predictions of the ball position correspond approximately to the true trajectory described, which suggests that they could be used for introducing predictive control.

The non-optimal case provides useful information for assessing the quality of the modelled dynamics. Since the measurement noise given by σ_v has been set much higher than the optimal case, during the filtering step of the Kalman filter the state update barely takes into account the information given by the new measurement. Instead, it relies more on the predictions given by the model dynamics. This is why the blue (filtered) trajectory is even smoother than in the optimal case.

In this sub optimal case, the average deviation between observed and filtered trajectories differs significantly, and the filtered position generally falls behind the measured one (slower motion), suggesting two things. First the modelled dynamics are not exactly the true ones, which is expected since the moments that the table exerts on the ball have not been considered. Second, there can be an error on the assumed rotation of the eye respect to the ground, since it determines the direction of the gravity vector respect to the table coordinate frame, and this is crucial in determining the direction of the acceleration of the ball.

Regarding the performance of the program, it processes all operations with no graphical output at 2.9 ± 0.2 ms each frame, or 344 ± 24 Hz, which is much greater than the required 50 Hz.

5.3 Limitations

A few limitations have been recognised during the testing phase, some of them due to the simulation software and others originated by the algorithms developed.

- There exist some situations in which the table can get too far out of the field of view, or too close to the camera so it becomes clipped by the near clipping plane, and the algorithm does not have enough information (or it is of poor quality) to properly detect the vertices of the square. These situations have been recognised in the implementation of the program and a warning has been set when they are detected. They can be recognised during the identification of the vertices of the square, in which the criterion $|t_a| \cdot |t_b| < 1.5$ is computed. In the general case in which the shape of the table has been deformed due to the above mentioned conditions, this criterion does not hold true for enough vertices (four), so in that case the program identifies that there is an underlying problem with the image.

- The program is not able to properly detect the table under extreme perspective views in which the plane formed by the X and Y coordinates of the TRF is almost perpendicular to the camera plane. In these situations, since the black borders of the table occlude a significant part of the red area of the table, the shape being recognised is far from the original square, and therefore, the solution for the PnP problem will not return reliable values. These extreme cases have not been found in the recorded video used, and are not likely to occur in normal operating conditions. However, if necessary, it could be solved by implementing an improved table detection algorithm that also takes into account the black borders and not only the red area of the table.
- As it has been mentioned before, the modelled dynamics are far from complete and accurate. This can be addressed by introducing information about the control actions and the table dynamics, and incorporating the orientation and position of the table as additional states in the linear model which the Kalman filter would also update and predict.
- The orientation of the camera respect to the ground has been deduced from observations and is not exactly the true one. This directly affects the estimated direction of the gravity force on the ball, so the true values would be needed in order to provide better estimation of such force direction.

6 Conclusions and future work

6.1 Conclusions

A program able to detect and track the position of the ball respect to the center of the table which is being observed under a changing perspective has been implemented.

The detection algorithm has shown to work under normal operating conditions, being able to recover the world position and orientation of the table and the position of the ball, and providing a signal indicative of the situations in which the quality of the table image is not enough to properly recover its location in the world.

A position tracking algorithm based on a Kalman filter has been successfully implemented. It provides the program with position estimates that effectively remove the noise present in the direct measurements given by the detection algorithm, and improves the quality by introducing a dynamics model and the physical constraints of the ball in the table.

The program has shown to be efficient by running all the operations at a much higher frequency than the rate of frame acquisition, indicating that it would be suitable to implement it in the simulation and process the information in real time.

6.2 Future work

Regarding the future work based on the algorithms developed and the experimental results, a few suggestions can be noted.

As stated before, the algorithm for detecting the table can be improved so it provides reliable values under extreme perspective cases. This can be done by introducing the detection of the table's black borders.

Regarding the rotation of the camera with respect to the ground, it would be required have this information for every frame of the simulation, so the direction of the gravity force with respect to the table can be computed for all circumstances in which the robot moves the eye's position and orientation.

Also referring to the data provided at every frame of the simulation, it would be useful to have the true position of the ball and orientation of the table to validate their estimates obtained from the program.

In order to improve the Kalman filter output, a more complete model is suggested in which the dynamics of the table are included, and the orientation and position of the table are introduced as state variables. To help with this regard, a more advanced version of a Kalman filter could be implemented. Particularly, the method based on learning the Kalman filter parameters with the aid of a Long Short-Term Memory (LSTM) artificial neural network [12] could be useful in the case in which a non-linear time variant model is required. This corresponds to the current project since the dynamics of the table and ball change over time, and the collision of the ball with the table represents a non-linearity.

Finally, the program can also be implemented inside the Neurorobotics Platform to further test in real time the algorithms under more operating conditions and to directly obtain the data above mentioned (rotation of the eye, table, and position of ball).

References

[1] I. Baira* Ojeda, S. Tolu*, H. H. Lund (2017). "A Scalable Neuro-inspired Robot Controller Integrating a Machine Learning Algorithm and a Spiking Cerebellar-Like Network". In: Mangan M., Cutkosky M., Mura A., Verschure P., Prescott T., Lepora N. (eds) Biomimetic and Biohybrid Systems. Stanford University, California, Living Machines 2017, July 26-28. Lecture Notes in Computer Science, vol. 10384, pp. 375-386. Springer. (* = equal contributions).

[2] Kalman RE (1960). "A New Approach to Linear Filtering and Prediction Problems". ASME. J. Basic Eng.;82(1):35-45. doi:10.1115/1.3662552

[3] Del Moral, Pierre (1996). "Non Linear Filtering: Interacting Particle Solution" (PDF). Markov Processes and Related Fields.

[4] Xiao-Shan Gao, Xiao-Rong Hou, Jianliang Tang, and Hang-Fei Cheng (2003). "Complete solution classification for the perspective-three-point problem". Pattern Analysis and Machine Intelligence, IEEE Transactions on, 25(8):930–943.

[5] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua (2009). "Epnp: An accurate o (n) solution to the pnp problem". International journal of computer vision, 81(2):155–166.

[6] Tong Ke and Stergios Roumeliotis (2017). "An efficient algebraic solution to the perspective-three-point problem". In Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on. IEEE, 2017.

[7] Levenberg, Kenneth (1944). "A Method for the Solution of Certain Non-Linear Problems in Least Squares". Quarterly of Applied Mathematics. 2: 164–168

[8] Hough, P.V.C (Dec. 18, 1962). "Method and means for recognizing complex patterns", U.S. Patent 3,069,654,

[9] Von Gioi, R. Grompone, et al (2010). LSD: "A fast line segment detector with a false detection control", IEEE Transactions on Pattern Analysis and Machine Intelligence 32.4: 722-732.

[10] Weisstein, Eric W. "Line-Line Intersection." From MathWorld--A Wolfram Web Resource. http://mathworld.wolfram.com/Line-LineIntersection.html

[11] Antonio, Franklin (1992). "Chapter IV.6: Faster Line Segment Intersection". In Kirk, David. Graphics Gems III. Academic Press, Inc. pp. 199–202. ISBN 0-12-059756-X

[12] Coskun, Huseyin et al (2017). “Long Short-Term Memory Kalman Filters: Recurrent Neural Estimators for Pose Regularization.” 2017 IEEE International Conference on Computer Vision (ICCV): n. pag. Crossref. Web.