Final Proyect - Supervised Learnig Solution

Enrique Chi Gongora, *1909040*, Jose Pat Ramírez, *2009104*, Rafael Marí Reyna, *2009083*,
Jesus Gabriel Canul, *1909031*, Hernando Te Bencomo, *2009132*, Raymundo Baas Cabañas, *1909008*,

***Abstract*—The project explores the use of supervised learning for automatic object identification using artificially generated ultrasonic sensor data. Various scenarios are simulated to capture a wide spectrum of sensor-object interactions. The data is further processed, allowing the training and evaluation of a predictive model. Model validation is performed through tests focused on its applicability in realistic environments, aiming at its integration in advanced robotic systems.**

I. Introduction

Object detection through supervised learning represents a dynamic and challenging field in artificial intelligence, with applications transforming robotics and automation. Our project addresses this challenge by developing a machine learning model capable of identifying the presence of objects using simulated data from ultrasonic sensors. With an innovative approach to data generation, our algorithm simulates realistic and complex scenarios, taking into account variables such as position, energy consumed and environmental conditions. The resulting dataset is diverse and contains multiple dimensions of information, creating a strong basis for training a model capable of detecting subtle variations in objects. Through preprocessing and the application of modeling techniques, we aim to create a versatile system for integration into robotics applications, contributing to the improvement of machine-environment interaction on multiple platforms.

To understand how the project was developed, let's understand the main part of it. Supervised learning is a fundamental approach in machine learning, where the goal is to teach computers how to perform tasks based on examples. It is similar to a learning process where someone studies with a teacher for example. Here, the teacher is the set of data provided to the algorithm, consisting of input-output pairs. The computer uses this data to learn how to map inputs to outputs. In practical terms, supervised learning is widely used for tasks like classification (where the output is a category) and regression (where the output is a numerical value). For instance, in classification, an email could be categorized as spam or not spam, and in regression, it might predict the price of a house based on its features like size and location.

The process of supervised learning has two main phases: training and testing. During training, the algorithm is exposed to a large dataset, learning to understand the relationship between the input data and their corresponding labels. In testing, the algorithm is given new data to see how well it can apply what it learned to make accurate predictions.

II. Methods and Tools

The methodology employed in this project involves a multifaceted approach to data generation and processing, as well as the selection and training of the most appropriate machine learning model. Initially, sensory data is simulated through specialized Python functions, such as simulate_ultrasonic_reading, which creates realistic distance readings based on the relative position and presence of objects. This simulation is further enhanced by simulate_motion and simulate_other_features, which add motion dynamics and other variables such as velocity and ambient temperature. These functions are explained in detail in the appendices of the document.

The simulated data are stored in a CSV file, and a copy of this is worked with to preserve the integrity of the original data set. Data cleaning and transformation of categorical variables into numerical variables is performed using the pandas library. Subsequently, the dataset is divided into training and test subsets using the scikit-learn tool.

After a comparative evaluation of several models, the Support Vector Machine (SVM) model was chosen for its higher performance. SVM is a powerful and versatile supervised learning algorithm whose main idea is to find a way to separate different categories of data points with a clear gap that's as wide as possible, using what is called in machine learning a hyperplane. In two dimensions, this hyperplane is a line, but in more complex datasets with many features, it can be a multi-dimensional surface.

When it comes to tuning an SVM, there are a couple of key parameters that need careful adjustment. The most important are the choice of the kernel function and the regularization parameter. The kernel function defines the type of hyperplane and transformation used for separation, while the regularization parameter controls the trade-off between having a wide margin and correctly classifying training data. A higher value of this hyperparameter tries to classify all training examples correctly, which can lead to overfitting, while a lower value results in a wider margin, possibly with some misclassifications in the training set.

For those reasons, these hyperparameters such as the kernel and the CC regularization parameter were optimized, seeking to improve the accuracy of the model. Several tests were performed, the results of which were measured with accuracy metrics and visualized through confusion matrices.

For the visualization of the results, graphical libraries such as matplotlib and seaborn were used. Seaborn is a Python data visualization library based on Matplotlib that offers a high-level interface for drawing attractive and informative statistical graphics. It is particularly suited for making complex plots more accessible. The robot trajectory was illustrated along

with object detection predictions, providing a clear graphical representation of the model's ability to identify obstacles. The accuracy of the model and its interpretation were significantly improved by hyperparameter tuning of the SVM, which was identified as the most effective model among those tested. The entire process from data generation to visualization of the results is documented in the appendices to provide a complete guide to the methodological approach adopted in the project.

III. Results

The results of the object detection model using a Support Vector Machine (SVM) indicate moderate performance with an accuracy of 56%. This accuracy metric is calculated as the number of correct predictions divided by the total predictions made. An accuracy of 56% suggests that the model is slightly better than a random choice, which would have an expected accuracy of 50% in a binary classification problem such as this, but there is certainly room for improvement.

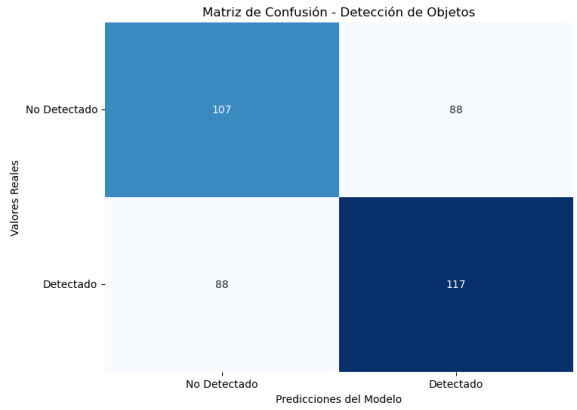

Fig. 1. Confusion Matrix

The matrix shows that the model correctly predicted the absence of an object (True Negatives) 107 times and the presence of an object (True Positives) 117 times. However, there were 88 False Positives, where the model erroneously predicted the presence of an object, and 88 False Negatives, where the model failed to detect an object that was present.

Visually, this is reflected in the graph of the robot trajectory with obstacle predictions. The points marked with a red 'x' represent locations where the model predicted the presence of an obstacle, and the blue line shows the robot's trajectory. The scatter of red 'x's relative to the blue trajectory provides a visual intuition of where and how many times the model perceived obstacles.

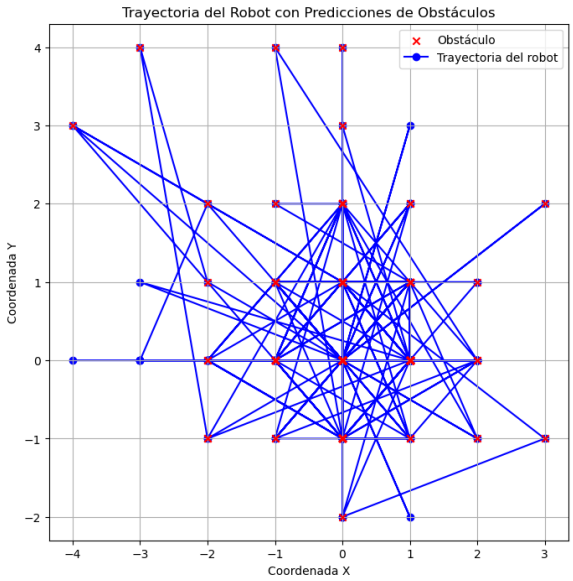

Fig. 2. Trajectory Simulation

The accuracy of the model is a starting point, but for practical applications, especially in robotics where wrong decisions can have significant consequences, it would be vital to improve this model. This could include collecting more data, testing different types of models, tuning hyperparameters, or employing feature engineering techniques to improve the model's ability to generalize from training data and make more accurate predictions on unseen data.

IV. Discussion

Discussion of the results indicates that the model has a level of accuracy that requires improvement to be effective in practical applications. With an accuracy of 56%, the model does not demonstrate sufficient reliability for real-world applications where accurate distinction between the presence and absence of objects is critical. The confusion matrix shows a considerable number of errors in both object detection and non-detection, implying that the model is often wrong in predicting.

These results are important because they point to areas where the project could be developed further. For example, the data being used to train the model could be reviewed and adjusted, or the use of different methods to build the model that might work better with this type of data could be explored. The discussion focuses on understanding how well the current model works and what steps could be taken to make it more accurate and reliable in the future.

V. Conclusion

In this study, we have presented an analysis of a predictive model for obstacle detection in the context of robot trajectory planning. Through the visual representation of results on the trajectory graph, where red 'x' markers denote obstacle predictions and the blue line represents the robot's path, we have comprehensively assessed the model's performance.

Our analysis reveals that, despite serving as a promising starting point, the model exhibits a substantial room for

improvement. The model's accuracy, standing at 56%, falls short of the reliability requirements for practical applications, particularly in the realm of robotics, where incorrect decisions can bear significant consequences. The confusion matrix underscores the existence of substantial errors in both object detection and non-detection, indicating that the model frequently makes erroneous predictions.

These results are of importance as they distinctly point to areas for future development in this project. It is evident that multiple approaches are available for enhancing the model. This may involve a thorough review of training data, exploration of alternative modeling methods better suited to this data, as well as fine-tuning of hyperparameters and feature engineering techniques. Our discussion has focused on understanding the current model's performance and outlining the necessary steps to increase its accuracy and reliability in practical applications.

In summary, our research provides a valuable foundation for future advancements in the field of obstacle detection in robot trajectory planning. We emphasize the urgent need to enhance the model's accuracy and have delineated potential strategies to achieve this goal. Future success in this area will depend on dedication and continuous innovation in the pursuit of solutions that enable robots to navigate more safely and efficiently in dynamic and challenging environments.

Appendix A
Simulated Data Generation

```
def simular_lectura_ultrasonico(posicion,
    objeto_presente):

    if 'contador_lecturas' not in
        simular_lectura_ultrasonico.__dict__:
        simular_lectura_ultrasonico.contador_lecturas =
        0

    if objeto_presente and
        simular_lectura_ultrasonico.contador_lecturas <
        40:
        simular_lectura_ultrasonico.contador_lecturas +=
        1
        distancia_base =
        np.linalg.norm(posicion)
        variabilidad =
        random.uniform(0, 1) * 5
        distancia_objeto =
        distancia_base +
        variabilidad
    else:
        distancia_objeto = 0

    return distancia_objeto
simular_lectura_ultrasonico.contador_lecturas =
0

def simular_movimiento():
    direccion = random.uniform(0, 2 *
    np.pi)
    distancia = random.uniform(0, 2)
    movimiento = {'direccion':
    direccion, 'distancia':
    distancia}
    return movimiento

def simular_otras_caracteristicas():
    velocidad_lineal =
    random.uniform(0, 1)
    velocidad_angular =
    random.uniform(0, 1)
    aceleracion_lineal =
    random.uniform(0, 1)
    aceleracion_angular =
    random.uniform(0, 1)
    tiempo = random.uniform(0, 1)
    energia_consumida =
    random.uniform(0, 1)
    temperatura_ambiente =
    random.uniform(0, 1)
    nivel_bateria = random.uniform(0,
    1)

    return velocidad_lineal,
    velocidad_angular,
    aceleracion_lineal,
    aceleracion_angular, tiempo, \
        energia_consumida,
        temperatura_ambiente,
        nivel_bateria

def generar_dataset(num_muestras,
    csv_filename="datos_robot.csv"):
    datos = []
    posicion_actual = np.array([0, 0])

    with open(csv_filename, 'w',
    newline='') as csvfile:
        fieldnames = ['x', 'y',
        'objeto_presente',
        'lectura_ultrasonico',
        'movimiento_direccion',
        'movimiento_distancia',
                      'velocidad_linea
                      l',
                      'velocidad_a
                      ngular',
                      'aceleracion
                      _lineal',
                      'aceleracion
                      _angular',
```

```
                'energia_consumi
                ↪  da',
                ↪  'temperatura
                ↪  _ambiente',
                ↪  'nivel_bater
                ↪  ia']
writer =
↪  csv.DictWriter(csvfile,
↪  fieldnames=fieldnames)

writer.writeheader()

for _ in range(num_muestras):
    movimiento =
    ↪  simular_movimiento()
    posicion_actual[0] += movi
    ↪  miento['distancia'] *
    ↪  np.cos(movimiento['dir
    ↪  eccion'])
    posicion_actual[1] += movi
    ↪  miento['distancia'] *
    ↪  np.sin(movimiento['dir
    ↪  eccion'])
    objeto_presente =
    ↪  random.choice([True,
    ↪  False])

    lectura_ultrasonico =
    ↪  simular_lectura_ultras
    ↪  onico(posicion_actual,
    ↪  objeto_presente)
    (velocidad_lineal,
    ↪  velocidad_angular,
    ↪  aceleracion_lineal,
    ↪  aceleracion_angular,
    tiempo,
     ↪  energia_consumida,
     ↪  temperatura_ambiente,
     ↪  nivel_bateria) =
     ↪  simular_otras_caracte
     ↪  risticas()
    datos.append({
        'x':
        ↪  posicion_actual[0],
        'y':
        ↪  posicion_actual[1],
        'objeto_presente':
        ↪  objeto_presente,
        'lectura_ultrasonico':
        ↪  lectura_ultrasonic
        ↪  o,
        'movimiento_direccion'
        ↪  :
        ↪  movimiento['direcc
        ↪  ion'],
        'movimiento_distancia'
        ↪  :
        ↪  movimiento['distan
        ↪  cia'],
        'velocidad_lineal':
        ↪  velocidad_lineal,
        'velocidad_angular':
        ↪  velocidad_angular,
        'aceleracion_lineal':
        ↪  aceleracion_lineal,
        'aceleracion_angular':
        ↪  aceleracion_angula
        ↪  r,
        'tiempo': tiempo,
        'energia_consumida':
        ↪  energia_consumida,
        'temperatura_ambiente'
        ↪  :
        ↪  temperatura_ambien
        ↪  te,
        'nivel_bateria':
        ↪  nivel_bateria
    })

    writer.writerow({
        'x':
        ↪  posicion_actual[0],
        'y':
        ↪  posicion_actual[1],
        'objeto_presente':
        ↪  objeto_presente,
        'lectura_ultrasonico':
        ↪  lectura_ultrasonic
        ↪  o,
        'movimiento_direccion'
        ↪  :
        ↪  movimiento['direcc
        ↪  ion'],
        'movimiento_distancia'
        ↪  :
        ↪  movimiento['distan
        ↪  cia'],
        'velocidad_lineal':
        ↪  velocidad_lineal,
        'velocidad_angular':
        ↪  velocidad_angular,
        'aceleracion_lineal':
        ↪  aceleracion_lineal,
        'aceleracion_angular':
        ↪  aceleracion_angula
        ↪  r,
        'tiempo': tiempo,
        'energia_consumida':
        ↪  energia_consumida,
        'temperatura_ambiente'
        ↪  :
        ↪  temperatura_ambien
        ↪  te,
        'nivel_bateria':
        ↪  nivel_bateria
    })
```

```
    print(f"El conjunto de datos ha
    ↪  sido guardado en
    ↪  '{csv_filename}'.")
    return datos

num_observaciones = 2000
conjunto_datos =
↪  generar_dataset(num_observaciones)
for i, observacion in
↪  enumerate(conjunto_datos):
    print(f"Observacin {i+1}:
    ↪  {observacion}")
```

APPENDIX B
DATA PROCESSING AND MODEL ANALYSIS

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
import seaborn as sns
```

```
df = pd.read_csv('datos_robot_copy.csv')
df
```

```
print("Valores nulos por columna:\n",
↪  df.isnull().sum())
```

```
print("Columnas con valores
↪  categóricos:\n",
↪  df.select_dtypes(include='object').
columns)
```

```
from sklearn.preprocessing import
↪  LabelEncoder

label_encoder = LabelEncoder()

df['objeto_presente_si']
↪  =label_encoder.fit_transform
(df['objeto_presente'])

df[['objeto_presente',
↪  'objeto_presente_si']].head()
```

```
print("Cantidad de valores igual a 1 en
↪  objeto_presente_si:",
↪  df['objeto_presente_si'].sum())
```

```
columnas_a_eliminar = ['objeto_presente',
↪  'energia_consumida',
↪  'aceleracion_angular',]
df = df.drop(columns=columnas_a_eliminar)
df.head()
```

```
y = df['objeto_presente_si']
X = df.drop(columns=
['objeto_presente_si'])
```

```
from sklearn.model_selection import
↪  train_test_split
from sklearn.svm import SVC
from sklearn.metrics import
↪  accuracy_score, confusion_matrix

X_train, X_test, y_train, y_test =
↪  train_test_split(X, y, test_size=0.2,
↪  random_state=42)

modelo_svm = SVC(kernel='linear', C=1.0)

modelo_svm.fit(X_train, y_train)

y_pred = modelo_svm.predict(X_test)

precision = accuracy_score(y_test, y_pred)
print(f"Precisión del modelo:
↪  {precision}")
```

```
matriz_confusion =
↪  confusion_matrix(y_test, y_pred)

fig, ax = plt.subplots(figsize=(8, 6))

class_names = ['No Detectado',
↪  'Detectado']

sns.heatmap(matriz_confusion, annot=True,
↪  fmt='d', cmap='Blues', cbar=False,
↪  ax=ax,
            xticklabels=class_names,
            ↪  yticklabels=class_names)

plt.title('Matriz de Confusión - Detección
↪  de Objetos')
plt.xlabel('Predicciones del Modelo')
plt.ylabel('Valores Reales')
plt.yticks(rotation=0)
plt.show()
```

```
plt.figure(figsize=(8, 8))
plt.scatter(X_test.loc[y_pred == 1, 'x'],
↪  X_test.loc[y_pred == 1, 'y'],
↪  marker='x', label='Obstáculo',
↪  color='red', zorder=2)
plt.plot(X_test['x'], X_test['y'],
↪  marker='o', label='Trayectoria del
↪  robot', color='blue', zorder=1)

plt.title('Trayectoria del Robot con
↪  Predicciones de Obstáculos')
plt.xlabel('Coordenada X')
plt.ylabel('Coordenada Y')
plt.legend()
plt.grid(True)
plt.show()
```

APPENDIX C
REFERENCES

1) N. Bellotto, K. Burn, and S. Wermter, “Appearance-based localization for mobile robots using digital zoom and visual compass,” *Robot. Auton. Syst.*, vol. 56, pp. 143–156, 2008. DOI: https://doi.org/10.1016/j.robot.2007.09.004

2) K. Y. Chan, R. Manduchi, and J. Coughlan, “Accessible spaces: Navigating through a marked environment with a camera phone,” in *Proceedings of the 9th International ACM SIGACCESS Conference on Computers and Accessibility*, Tempe, AZ, USA, 14–17 October 2007.
3) “DOF a entidades de obstáculo (Inteligencia)—ArcGIS Pro — Documentación,” Accedido el 13 de noviembre de 2023. En línea. Disponible: https://pro.arcgis.com/es/pro-app/3.0/tool-reference/intelligence/dof-to-obstacle-features.htm
4) “Obstacle Avoidance Robot Object Detection Dataset (v1, 2023-07-13 2:05pm) by Moratuwa Project,” Roboflow. Accedido el 13 de noviembre de 2023. En línea. Disponible: https://universe.roboflow.com/moratuwa-project/obstacle-avoidance-robot/dataset/1
5) E. Yi Kim, “Wheelchair Navigation System for Disabled and Elderly People,” *empirica*, 2016.