Navegación autónoma mediante aprendizaje por refuerzo

Ignacio Agustín Costa[1], Esteban José De Leo[1], Iris Inés Sattolo[1], Marisa Daniela Panizzi[1]
Escuela Superior de Ingeniería, Informática y Ciencias Agroalimentarias. Universidad de Morón. Cabildo 134 (B1708JPD). Partido de Morón, Argentina.
costa.nacho@gmail.com; deleoesteban@gmail.com; iris.sattolo@gmail.com; marisapanizzi@outlook.com

Resumen. Existen diversas situaciones, como derrumbes edilicias o catástrofes, en donde seres vivos quedan atrapados y su supervivencia depende de encontrar el camino de salida del lugar. Este trabajo presenta el diseño e implementación de un sistema de aprendizaje mediante la utilización de Redes Neuronales Artificiales (RNA) con aprendizaje por refuerzo, con el propósito de resolver laberintos de estructura desconocida con un robot autónomo (RA). Se definieron un conjunto de políticas y puntajes para llevar acabo el aprendizaje por parte del agente, y posteriormente encontrar la solución buscada. La RNA fue implementada sobre las librerías Tensorflow y Keras. El RA es comandado por Arduino, el cual se comunica con una aplicación de escritorio. Finalmente, se diseñó una maqueta adaptable a cualquier laberinto para poder ejecutar la resolución mediante el RA. Los resultados indican que el aprendizaje por refuerzo es apropiado para resolver esta problemática, dado que es adaptable a problemas complejos.

Palabras clave: aprendizaje por refuerzo, red neuronal artificial, RA, q-learning, resolución de laberintos, exploración de estructuras desconocidas.

1 Introducción

Actualmente, existes situaciones y escenarios reales donde el ambiente es desconocido para los individuos que se encuentran en él, y es imperativo encontrar una salida. Algunas situaciones pueden ser catástrofes naturales o derrumbes edilicios, en donde seres vivos quedan atrapados entre restos o escombros, y su supervivencia depende de encontrar el camino de salida del lugar.

Existen diversos algoritmos que proporcionan respuestas acordes al problema de encontrar el trayecto óptimo dentro de un ambiente desconocido, algunos de ellos son: el algoritmo estrella [1] y doblar siempre a la izquierda/derecha. Los mismos, aplican procedimientos para encontrar un camino válido, dentro de todos los posibles. Es decir, están restringidos a estructuras simples en donde hay espacios físicos ocupados, y otros libres. La resolución al problema sólo recae en recorrer los espacios libres hasta encontrar una salida. El ejemplo más sencillo sería imaginar a una persona rodeada de

escombros resultantes de un derrumbe, en donde se desea encontrar un camino seguro de salida de la zona.

Pero, hay escenarios en donde aplican reglas más complejas. Por ejemplo, si la persona en medio del derrumbe debe rescatar a otros antes de encontrar la salida, estos algoritmos no nos servirían.

Otro caso sería, si un drone debe recorrer la zona de derrumbe para identificar los daños ocasionados. Según las características técnicas del dispositivo, podría elevarse por encima de obstáculos de cierta altura, pero no estructuras demasiado altas.

De acuerdo a los problemas mencionados, en esta investigación se propuso como solución crear un sistema de aprendizaje para la resolución de caminos en ambientes complejos de estructura desconocida, y su ejecución con un agente autónomo.

Se desarrolló una aplicación que permita ingresar los datos de entrada del entorno, realice el entrenamiento mediante una RNA y envíe la solución al RA mencionado.

Este artículo se organiza de la siguiente manera: se presentan los Trabajos relacionados (en la sección 2), el desarrollo de la solución (en la sección 3), su validación (en la sección 4) y, por último, las conclusiones y las futuras líneas de trabajo.

2 Trabajos relacionados

Se realizó un Estudio de Mapeo Sistemático (en inglés, Systematic Mapping Study o SMS) de acuerdo con el proceso propuesto en [2] para analizar el estado del arte sobre el aprendizaje automático para resolución de caminos de estructuras desconocidas. Por razones de espacio, en [3] se encuentra información sobre el SMS.

En este estudio se seleccionaron 28 estudios primarios de un conjunto inicial de 2450 artículos que arrojó la búsqueda. Estos se obtuvieron realizando una búsqueda automática en las librerías digitales Google Scholar, IEEE Xplore y Scielo, en el período comprendido entre el año 2008 y 2018.

Una vez analizados los estudios primarios, se concluyó que:

- El 50% de las publicaciones seleccionadas (14 artículos), hacen uso de algoritmos de resolución de caminos los cuáles no realizan un procedimiento de aprendizaje. En cambio, sólo el 14% de las publicaciones utilizan modelos de aprendizaje.
- Dentro de los modelos de inteligencia artificial, el algoritmo genético, las RNA tipo SOM y convolucionales fueron los predilectos, ocupando un 8%, 4% y 4%, respectivamente.
- Un 40% de las publicaciones que construyeron RA (20 publicaciones) optaron por hacer un diseño propio de la electrónica del RA. El restante 60% se volcó a marcas reconocidas de microcontroladores. Las más populares fueron Arduino con un total de 4 artículos (20%) y en segundo lugar Pioneer con 3 artículos (15%).
- Como lenguajes predilectos se aprecia una amplia proporción de uso de Matlab y C++, siendo un 33% cada uno. Ambos lenguajes, de bajo nivel, permiten realizar cálculos específicos para realizar estas operaciones. Relacionado con los resultados de la PI3, los RA desarrollados con la placa Raspberry Pi, utilizaron Python. En contraparte, los RA desarrollados con la placa Arduino, utilizan C++ como lenguaje.

- Ninguno de los RA sigue un recorrido preestablecido, se encontró que todos los modelos diseñados avanzan hasta encontrarse con un obstáculo, para luego moverse en una dirección sin destino determinado. Esto se hace hasta toparse con la solución.
- La mayor cantidad de fuentes de congresos fue obtenida de Argentina y España, con un resultado de 4 cada uno. Con respecto a las publicaciones de revistas, Colombia y Ecuador fueron mayoría, con un total de 4 y 3 respectivamente.

3 Desarrollo de la solución

Para la construcción de la solución se definieron los siguientes componentes:

- Maqueta.
- Aplicación de escritorio.
- Robot Autónomo.

En [3] se describe con un mayor nivel de detalle, la maqueta del laberinto y la aplicación de escritorio.

El ambiente a recorrer será representado por una maqueta a escala. La misma consta de un papel subdividido en cuadrados que representa una sección del terreno. Sobre el papel se ubicarán paredes de madera que delimitarán los caminos a recorrer.

El usuario debe plasmar el escenario real en la maqueta y en la aplicación de escritorio. La misma recibirá como input la estructura a recorrer. Una vez ingresada, el usuario inicia el proceso del entrenamiento. Durante el mismo, la aplicación muestra en pantalla los datos del entrenamiento. Una vez finalizado, se habilita la ejecución del RA para recorrer camino óptimo hacia la salida de la estructura.

El RA es el encargado de recorrer la maqueta. El mismo está diseñado con un microcontrolador Arduino y se comunica a través del protocolo Bluetooth con la aplicación de escritorio. El usuario inicia la ejecución luego del entrenamiento, el RA recibe la información y recorre la estructura hasta la salida.

A continuación, se presenta el modelo de aprendizaje (sección 3.1.) y la implementación de la solución (sección 3.2.).

3.1 Modelo de aprendizaje

El modelo de aprendizaje utilizado es el aprendizaje por refuerzo. El mismo propone obtener recompensas por cada acción realizada pero la retroalimentación no es constante, dado que un sólo movimiento no resolverá el problema. Será un conjunto de los mismos, el que recompense al agente eventualmente. Basado sólo en esas recompensas, el agente tiene que aprender a comportarse en el medio ambiente. No se le dice qué acciones tomar, si no que él debe experimentar para encontrar qué acciones lo llevan a una mayor recompensa. Los casos más desafiantes son los que no llevan a una recompensa inmediata, si no en las siguientes situaciones.

Se considera modelar la resolución del laberinto, como un Proceso de Decisión de Markov [4]. En donde cada estado se verá constituido por la estructura a recorrer y la posición del agente. La transición entre estados se dará a partir de un movimiento del

agente en el entorno. Esto resultará en una recompensa positiva o negativa, cuantificada por un número entero.

Q-Learning es la técnica de aprendizaje por refuerzo que será utilizada. La misma establece la necesidad de diseñar una política que defina el modo en que el agente se comporta en un momento dado. El objetivo es que el agente aprenda de dicha política. Cada par estado-acción resultará en un valor "Q" (derivado de la palabra inglesa "quality"), que representa la recompensa futura máxima [5].

El objetivo de Q-Learning es desarrollar una política que maximice las recompensas que se obtienen al ejecutar una acción, esto es, tomar la mejor decisión en un momento y contexto dado. Se denomina Explotación, a tomar decisiones en base a lo aprendido hasta el momento por el algoritmo. De esta manera se conoce el resultado esperado, y se refuerzan estas decisiones [6].

En contraposición, la Exploración se refiere a tomar decisiones que normalmente no tomaríamos, con la posibilidad de ganar nuevo conocimiento [7].

Como motor de aprendizaje y almacenamiento del conocimiento, se utilizará una RNA Prealimentada (feed-forward en inglés). La misma, recibirá como entrada la estructura del laberinto y la posición del RA. La salida serán los 4 movimientos posibles (arriba, abajo, izquierda y derecha) con sus probabilidades, según la predicción de la próxima acción.

3.2 Implementación de la solución

El laberinto será modelado como una matriz de números enteros. Cada celda correspondiente a la cuadrilla tendrá un valor asociado representando su estado actual:

- 0 = Celda libre: Es aquella celda a la cual el RA puede ingresar.
- 1 = Celda ocupada: Es aquella celda a la cual el RA no puede ingresar.
- 99 = Celda del robot: Es la celda en la cual se encuentra el RA ubicado.

Para la representación del laberinto se creó una maqueta adaptable a cualquier diseño. Para el terreno se imprimió un plotter con la cuadrícula de 6 filas y 6 columnas (36 celdas en total).

Para representar las celdas ocupadas, se fabricaron paredes de madera de 3mm cortadas con láser para una mayor precisión.

Como interfaz con el usuario, se creó una aplicación de escritorio en Python 3.7 utilizando la librería gráfica Tkinter. Este programa tomará como input el laberinto provisto por el usuario y permitirá realizar el aprendizaje por parte del algoritmo. Durante el proceso de entrenamiento, se visualizarán datos estadísticos correspondientes. Luego de finalizado el entrenamiento, se permitirá acceder a reportes con métricas de los resultados del mismo. Asimismo, se mostrará una animación con la solución encontrada. Por último, el programa se comunicará a través de Bluetooth, utilizando la librería X, con el RA enviando la información de la resolución.

Para el entrenamiento, como parámetro de entrada necesitaremos la matriz del laberinto. Con las dimensiones de la estructura se crea la RNA y se iniciará con valores de pesos aleatorios.

Se define a un “estado” como una foto del laberinto en un momento determinado. La misma representa al entorno y la posición del RA. Llamaremos “transición” al cambio de un estado a otro, ante una acción (movimiento de celda). Utilizaremos el modelo de Proceso de Decisión de Markov.

Se ejecutarán igual cantidad de corridas como epochs definidos, por defecto 1000. Llamamos una “corrida” a la serie completa de movimientos del RA desde la posición inicial hasta la salida o la penalización máxima. Para cada corrida, la posición inicial del RA será determinada de forma aleatoria por el algoritmo.

En cada corrida, se ejecutarán una cantidad de movimientos (acciones) a los cuales consideraremos episodios. En cada episodio, se evalúa si realizar explotación o exploración. Establecemos el coeficiente de exploración inicial al 10%. Al superar el 90% de corridas exitosas, se disminuye al 5%. Para explotar, se envía el estado a la RNA y se consulta el movimiento a realizar.

Cada episodio se almacenará en una memoria interna. El registro del episodio estará compuesto por cinco partes: Estado Inicial, Acción, Recompensa, Estado Final e indicador de finalización de corrida.

Para poder definir las reglas del problema y encontrar la solución requerida, se definió una política. La misma buscará maximizar la recompensa obtenida ante cada acción.

A través de un sistema de puntajes, la política le permitirá al RA aprender y así hallar un camino óptimo. Cada movimiento, obtendrá una recompensa positiva o negativa. La corrida comprenderá la suma de puntajes de todos los movimientos.

En la Tabla 1 se detallan las acciones junto con sus respectivos puntajes.

Tabla 1. Puntajes por acción**.**

Acción	Puntaje
Movimiento a una celda libre	-0.04
Movimiento a una celda ya visitada	-0.25
Encontrar la salida	1
Movimiento a una celda ocupada	-10
Comenzar la corrida en una celda bloqueada	Puntaje negativo límite
Movimiento fuera de los límites del laberinto	Puntaje negativo límite

Para alentar a realizar la mínima cantidad de movimientos posible, se penaliza con-0.04 cada ingreso a una celda libre.

Para alentar a no volver sobre sus propios pasos, se penaliza con -0.25 al volver a ingresar a una celda visitada previamente.

Al encontrar la salida, se obtiene +1, la cual es la máxima recompensa.

Si durante la corrida se llega al puntaje negativo límite, se considera como perdida la corrida, dado que el RA cometió demasiados errores. El cálculo del límite es: -0.5 * tamaño laberinto.

Para desalentar a ingresar a una celda que se encuentra ocupada, daremos la penalidad máxima igual al puntaje negativo límite. Este movimiento sólo se permite durante el entrenamiento.
Para desalentar a ir por fuera de los límites del laberinto, daremos la penalidad máxima igual al puntaje negativo límite. Este movimiento sólo se permite durante el entrenamiento. Al comienzo, el RA cometerá errores, de los cuales irá aprendiendo en pos de obtener mayores recompensas.

Luego de cada episodio, la RNA aprende en base a la recompensa obtenida. El entrenamiento continuará durante todas las corridas o hasta llegar a un nivel de confianza aceptable. Se determina que un nivel de aprendizaje es aceptable, si encuentra la salida del laberinto partiendo de cualquier celda libre.

Se utilizará una RNA prealimentada multicapa. Poseerá una capa oculta, siendo 3 en total. La entrada de la red será la matriz que representa al laberinto convertida en un vector. Esto quiere decir, que la cantidad de nodos de la capa de entrada será igual a la cantidad de celdas. La capa oculta, tendrá el mismo tamaño que la capa de entrada. La capa de salida poseerá 4 nodos: uno por cada movimiento posible (arriba, abajo, izquierda y derecha). El resultado de cada nodo será el valor Q estimado para ese contexto y movimiento.

Como función de activación, utilizaremos la Función Sigmoide (S-Shaped function) [8]. De esta forma, podremos permitir a la red activar binariamente cada nodo.

Como optimizador de gradientes estocásticos, utilizaremos el método Adam. [8]

Como función de pérdida utilizaremos el Error Cuadrático Medio (Mean Squared Error) [8]. El mismo, es un estimador que mide el promedio de los errores al cuadrado.

La red será programada en el lenguaje Python 3.7.0. Se utilizaron las bibliotecas Keras 2.2.4 y Tensorflow 1.15.0, se construyó la red y se ejecutaron todas las operaciones matemáticas necesarias para predecir, aprender y calcular métricas de estos procesos.

En la Tabla 2 se presentan las características y componentes del RA, y en la Figura 1 el RA construido.

Tabla 2. Características y componentes del RA.

Componente	Cantidad	Detalle
Chasis	1	Estructura circular con dos niveles
Ruedas	3	Dos ruedas laterales de madera y una rueda tipo bola giratoria
Microcontrolador	1	Arduino Mega 2560
Motor	2	Motor “paso a paso” modelo 28BYJ
Controlador de Motor	2	Driver modelo ULN2003
Alimentación	2	Batería externa de 5V
Comunicación	1	Módulo Bluetooth HC-05

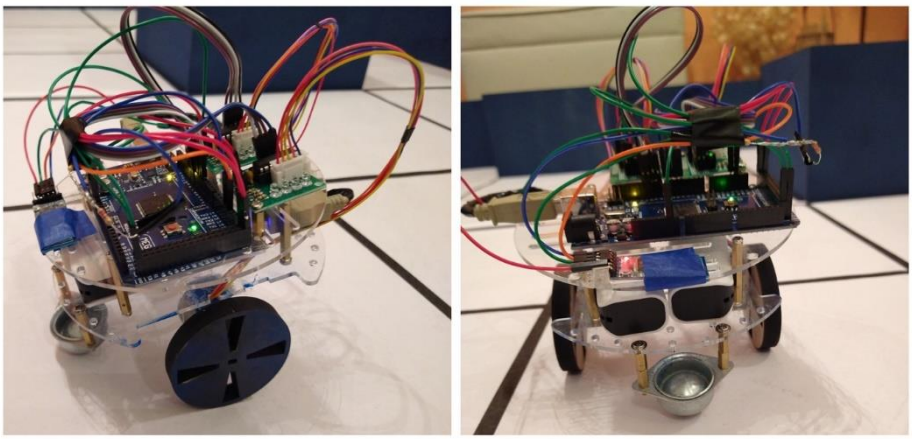

Fig. 1. Robot autónomo.

El sistema de navegación recibe la información de los movimientos a realizar y los ejecuta en el orden indicado.

El RA es capaz de realizar tres tipos de movimiento: Avanzar, Doblar a la derecha y Doblar a la izquierda.

Los motores paso a paso, como su nombre lo indica, dividen un giro completo del eje en 4096 pasos. Dicho esto, para ponerlos en funcionamiento se decide una dirección y una cantidad de pasos a ejecutar.

El movimiento de avanzar acciona los dos motores hacia delante y ejecuta 6400 pasos. Este movimiento contempla que el RA este posicionado en el medio de un cuadrado y finalice en el medio del cuadrado siguiente.

El movimiento de doblar a la derecha es un giro de 90° grados en sentido horario. Acciona el motor de la izquierda hacia delante y el de la derecha hacia atrás, ambos motores ejecutan 1800 pasos cada uno.

El movimiento de doblar a la izquierda es un giro de 90° grados en sentido antihorario. Acciona el motor de la derecha hacia delante y el de la izquierda hacia atrás, ambos motores ejecutan 1800 pasos cada uno.

4 Validación de la solución

Luego del diseño e implementación de la solución propuesta se ejecutaron diversas pruebas integradoras. Las mismas contemplaron distintos niveles de complejidad de aprendizaje, por medio de laberintos de diferentes diseños. A continuación, se presenta un caso de los tantos realizados.

4.1 Caso de prueba: Laberinto simple con diversos caminos posibles

Para este caso de prueba se diseñó un laberinto con algunas celdas ocupadas dispersas, pero con la mayoría de celdas libres. Esto genera múltiples caminos incluso combinables.

El objetivo de esta prueba es que el algoritmo entrene, buscando una solución para este laberinto. Como resultado esperado, deberá elegir el camino más óptimo, de entre los tantos caminos posibles.

Luego de la ejecución y posterior análisis de los resultados obtenidos, podemos concluir que:

- El entrenamiento finalizó en el epoch 65. Este desempeño es aceptable, dado que encontró la solución óptima a los 9 minutos.
- El algoritmo pudo elegir el camino más óptimo entre tantos posibles.
- El camino elegido contiene trayectos rectos, para evitar movimientos innecesarios.
- A partir del epoch 18 del entrenamiento, el algoritmo comenzó a encontrar posibles soluciones al laberinto.
- En el epoch número 30 del entrenamiento, se alcanzó un win rate del 50%, habiendo encontrado ya 10 soluciones posibles.
- Pasados 5 minutos de entrenamiento, comienza una crecida constante de nuevas soluciones encontradas.
- Al principio del entrenamiento, la función de pérdida devuelve valores altos. Con el transcurso del tiempo, el aprendizaje se ve reflejado en la notoria disminución de la función a partir del epoch 23.
- El RA demoró 40 segundos en recorrer el laberinto.

El diseño y la solución obtenida se pueden visualizar en la Figura 2.

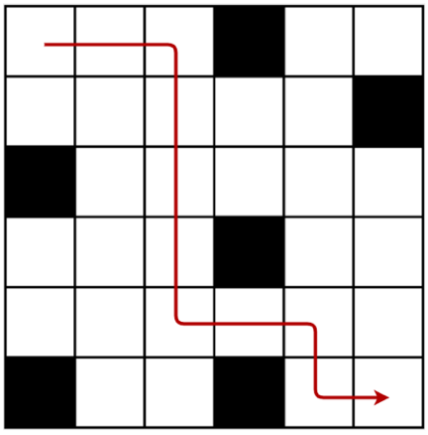

Fig. 2. Diseño y resolución del laberinto del primer caso de prueba.

5 Conclusiones y futuras líneas de trabajo

En este trabajo se propuso implementar un algoritmo que aprenda por refuerzo a resolver la problemática planteada, haciendo uso de un sistema de políticas y puntajes. Para modelar el problema se consideró la estructura como un laberinto y el agente junto con sus movimientos y estados como un proceso de decisión de Markov. Esto nos permitió considerar cada estado como la ubicación del agente en el escenario, y cada movimiento como una transición de estados.

Q-Learning fue la técnica de aprendizaje por refuerzo que se utilizó. La misma permitió definir una política que constituyó un conjunto de reglas que determinaron el accionar del agente durante el aprendizaje. Este aprendizaje es obtenido mediante un proceso de entrenamiento, el cual consiste en la búsqueda de una salida desde diferentes puntos de partida. Esto es ejecutado durante varias iteraciones y finalmente evaluado para comparar soluciones. Dicha evaluación está dada por la comparación de puntajes obtenidos en cada solución. El sistema de puntaje diseñado busca incentivar al agente a realizar movimientos válidos en búsqueda de la solución del camino óptimo, y desalentar movimientos inválidos como así también trayectorias innecesarias.

La implementación presentada es aplicable a cualquier escenario y adaptable a cualquier regla del entorno. Esto se puede llevar a cabo mediante la redefinición de políticas y puntajes, estableciendo una lógica de comportamiento.

Por otra parte, este trabajo de investigación no sólo estuvo enfocado a la resolución de estructuras desconocidas, sino que también a la ejecución de la solución por medio de un RA. La combinación del algoritmo de aprendizaje y su ejecución por medio del RA, demostró proveer una solución integral a la problemática con un alto grado de satisfacción. Mediante las pruebas de distintos tipos de motores y estructuras de autos, se pudo concluir que, teniendo un alto de grado de precisión en los mismos, son prescindibles los diversos sensores que dan soporte adicional a la navegación.

A pesar de los avances realizados en este trabajo, en cuanto a la navegación autónoma en estructuras desconocidas, aún quedan diversos desafíos por resolver. Estos desafíos contemplan desde etapas de reconocimiento del escenario hasta el diseño del RA adaptable a las necesidades del problema. A continuación, se enumeran las diferentes líneas de trabajo futuro surgidas de estos desafíos:

- Reconocimiento del escenario por parte del RA, creando un mapa del entorno que servirá como entrada al proceso de aprendizaje.
- Evaluación de las condiciones físicas del entorno para detectar posibles riesgos. Esto se puede llevar a cabo, incorporando diversos sensores como por ejemplo, detector de llama, sensor de temperatura, sensor de humo, etc.
- Transmisión de video y audio en tiempo real del escenario.
- Implementar un sistema de visión artificial con el fin de detectar posibles víctimas.
- Desarrollar un sistema escalable de coordinación de trabajo en conjunto entre varios RA para dividir tareas.
- Ampliar el rango de comunicación entre la aplicación y el RA para operar a mayor distancia.

6 Referencias

1. Cui, X., & Shi, H. (2011). *A*-based Pathfinding in Modern Computer Games*. IJCSNS International Journal of Computer Science and Network Security, 11(1), 125-130.
2. Kitchenham, B., Brereton, P., & Budgen, D. (2015). *Evidence-Based Software Engineering and Systematic Reviews.* USA: Chapman and Hall 1 st.Editon. Chapman and Hall/CRC.
3. Costa I., De Leo E., Panizzi M., Sattolo I. Anexo - Navegación autónoma mediante aprendizaje por refuerzo. https://doi.org/10.6084/m9.figshare.12676685.v1
4. Hamdy, T. (2004). *Investigacion de operaciones*. México: Pearson Educación.
5. Nazia, H. (2019). Hands-On Q-Learning with Python: *Practical Q-learning with OpenAI Gym, Keras and Tensorflow*. Reino Unido: Packt.
6. Sean, S., Yang, W., & Rajalingappaa, S. (2018). *Python Reinforcement Learning Projects: Eight hands-on projects exploring Reinforcement learning algorithms using TensorFlow*. Reino Unido: Packt.
7. Ameet, J. (2020). *Machine Learning and Artificial Intelligence. EEUU: Springer*.
8. Sharma, A., Ravi Vishwesh, S., & Beyeler, M. (2019). *Machine Learning for OpenCV 4: Intelligent algorithms for building image processing apps using OpenCV 4, Python, and scikit-learn*. Reino Unido: Packt.