Definir Viabilidad de Destino de manera costo-eficiente

Kaucic Florencia, Bistolfi Facundo Raul, Palopoli Juan José, Martinez Sebastian

Universidad Nacional de La Matanza,
Departamento de Ingeniería e Investigaciones Tecnológicas,
Florencio Varela 1903 - San Justo, Argentina
florkaucic@gmail.com, facundobistolfi@hotmail.com, juanjopalopoli@hotmail.com,
sebastianmartinez09@gmail.com

Resumen. La finalidad de este documento es determinar si es posible alcanzar el destino suministrado al CarryBot, de manera costo-eficiente, empleando Machine Learning, y por ende, HPC.

Palabras claves: CarryBot, HPC, Machine Learning, Obstaculos

1 Introducción

Previo a profundizar en lo que trata el documento, se deja en claro que el mismo esta dirigido al proyecto CarryBot[1]. El proposito del mismo, es profundizar en cuanto a determinar si el destino que se le suministra al CarryBot, es realmente alcanzable, mediante el uso de Machine Learning, y por consiguiente, HPC.

Si nos situamos en el hoy, Junio de 2018, es sabido que se han realizado grandes avances en cuanto a lo que refiere a la autonomia de los vehiculos. Un ejemplo claro de esto, es Waymo[2], empresa que originalmente empezo como un proyecto de Google, sobre un auto con conduccion autonoma.

Otra aplicación semejante, es LingoLens[3], que mediante el uso de Realidad Aumentada y Machine Learning, permite identificar los objetos, y presentar la palabra correspondiente a dicho objeto, en la lengua que el usuario quiere aprender, es decir, emplea dichas tecnologias para fomentar el aprendizaje bajo demanda (Pull-learning).

Citando lo dicho en el parrafo de esta introduccion, la finalidad de este modulo tiene origen en la necesidad de determinar, de la manera mas eficiente en cuanto al costo posible, si el CarryBot se encuentra encerrado, es decir, si los obstaculos aledaños le impediran alcanzar el Destino asignado.

[1] https://github.com/FlorKaucic/SOA-CarryBot
[2] https://waymo.com/
[3] http://anshulbhagi.com/projects/lingolens/

2 Desarrollo

Nuestra investigacion tiene origen en la necesidad de solucionar una interrogante: ¿Existe alguna manera de que el CarryBot pueda percatarse de que su destino es inalcanzable? Las respuestas a esta interrogante son diversas.

Antes de tratar el algoritmo en si, hay que comprender mas en detalle como trabaja CarryBot. Explicado de manera general, CarryBot recibe la direccion del modulo Bluetooth de destino, se obtiene un promedio de la intensidad (RSSI) del modulo de destino, y se desplaza. Luego de detenerse, comparando contra otra muestra de intensidad, se determina si continuar en la misma direccion o rotar, y se desplaza.

El proceso descripto previamente se repite una y otra vez, hasta alcanzar el destino o bien, hasta que se detiene por haber sido obstaculizado una cierta cantidad de veces. De esta manera, por prueba y error, CarryBot alcanza su destino, *siempre y cuando exista la posibilidad de alcanzarlo*.

Es por esto que, es necesario determinar si, desde una cierta posicion, es posible alcanzar el destino, ya que el costo de esquivar obstaculos conforme se rastrea el destino mediante la intensidad, en los peores casos puede ser altamente costoso en cuanto al consumo de energia y tiempo.

Para esto, se decide atacar el problema explotando el potencial que nos brinda Machine Learning, en conjunto con HPC, mas especificamente, GPU. Citando [3], una fuerte alternativa para implementarlo es con CUDA Kernels, por ejemplo NVI, lenguaje CUDA de NVIDIA. Cabe destacar que no es de nuestro interes identificar cada objeto presente, es decir, *determinar que es*, simplemente saber que *hay un obstaculo ahi*.

Otro aspecto a remarcar es que CarryBot esta equipado con un sensor ultrasonico, por lo que, para poder llevar a cabo lo antes mencionado, seria necesario incorporar otra pieza de Hardware. Acorde a lo indicado en [1], con una camara que posea una resolucion de 640x480, con la cual capturar imágenes en escala de grises de 8 bits de intensidad, sera suficiente para alcanzar la meta deseada.

Teniendo en cuenta lo dicho hasta este punto, la solucion propuesta seria hacer rotar 360º el CarryBot, con pequeños intervalos en los que se detendria, para tomar una imagen. Una vez realizada la vuelta completa, se analizaria el lote de imágenes para determinar la profundidad del area, y los posibles caminos a seguir. Este proceso se repetiria hasta alcanzar el destino o bien, determinar que se encuentra encerrado.

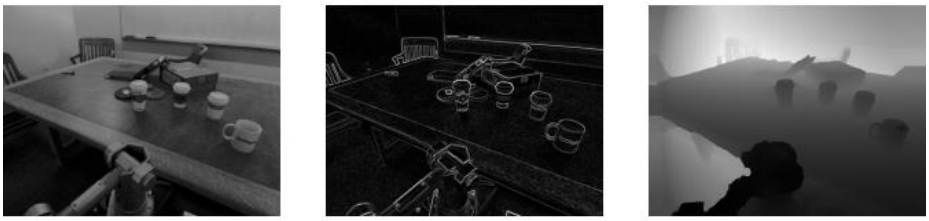
Queda claro bajo lo anterior que el CarryBot podria verse atrapado en un loop infinito si cada vez que se encuentra con que no tiene salida retrocede, por lo cual, es de nuestro interes para solucionar esto, registrar los desplazamientos que se van realizando para formar un mapa. De esta manera, aprovechando el alto rendimiento que nos brinda GPU, se podria determinar sin mayor perdida, si se encuentra dando circulos, es decir, verdaderamente encerrado, y en tal caso, detenerse ya que es imposible alcanzar el destino.

3 Explicación del algoritmo.

```
function anguloSiguiente(mapa)
#
        // angulo al que girar para continuar hacia una zona no recorrida
        angulo = mapa.sinrecorrer(posX, posY)
        if angulo == NULL
                return FIN
        else
                return angulo
#

function obtenerImagenes(cantidadImagenes)
#
        for( i = 1 ; i <= cantidadImagenes; i++)
        #
                lista.agregar( camara.capturar() )
                motores.rotar(90) // vehiculo rota 90 grados
        #

        return lista
#

function actualizarMapa(mapa, imagenes)
#
        // clase que maneja todo el apartado HPC
        hpcengine.clear()
        hpcengine.analizar(imagenes)
        hpcengine.actualizar(mapa)
#

/* Principal */
destinoAlcanzado = en_destino()
angulo = anguloSiguiente(mapa)

while angulo <> FIN && destinoAlcanzado == FALSE
#
        imagenes = obtenerImagenes(4)
        actualizarMapa(mapa, imagenes)

        // avanza con el angulo indicado
        avanzar(angulo)
        destinoAlcanzado = en_destino()
        angulo = anguloSiguiente(mapa)
#
```

Si bien el algoritmo enunciado previamente deja en claros varios rasgos del Desarrollo realizado, hay diversos aspectos que quiza no quedan en claro.

En principio, la rotación de 90º de la que se habla en *obtenerImagenes* es producto de que, para producir una vista panorámica, es suficiente con 4 imágenes.



(a) (b) (c)

Ilustración 1: Imágenes de ejemplo tomadas de [1] donde se puede observar (a) intensidad en escala de grises (b) gradiente de intensidad (c) profundidad

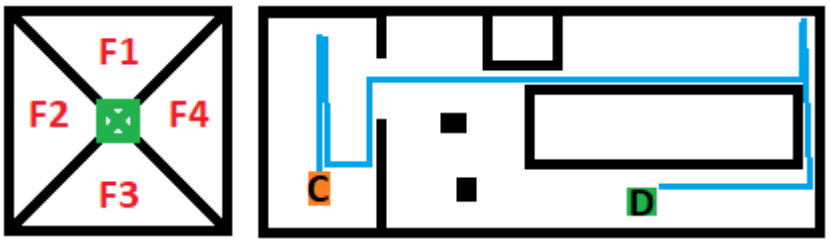


Ilustración 2: Se puede observar a la izquierda, el concepto panorámico con 4 imágenes, y a la derecha, un ejemplo visual del mapa, en naranja la ubicación inicial de CarryBot, en verde el Destino, y en celeste el trayecto recorrido.

4 Pruebas que pueden realizarse

Las pruebas que pueden realizarse son variadas, aunque muchas de ellas giran en torno a conceptos similares entre si. Independientemente de esto, hay una serie de pruebas basicas que deberian de hacerse para poder poner a prueba la propuesta.

Como primera prueba, colocar el CarryBot en un area cerrada, sea esta circular o cuadrada. Si bien es una prueba basica, sirve como pie inicial para medir el nivel de progreso del desarrollo, ya que es un caso donde el Destino no es alcanzable, y el entorno en el que el dispositivo se desplazara no es complejo. Esto permitiria analizar cuanto tiempo demora en percatarse de que se encuentra encerrado y analizar cuanta energia consumio para ello.

Como siguiente prueba, colocar el CarryBot a varias secciones/habitaciones de distancia del Destino, pero pudiendo este ser alcanzado. Esto permitira evaluar diversos factores como: El tiempo que demora para alcanzar el objetivo, que ruta adopta en cada variante de esta prueba, el consumo realizado, entre otros.

5 Conclusiones

En resumen, hemos planteado una propuesta en la cual, empleando una camara fotografica, y otras tecnologias como HPC, especificamente GPU, se pueden mejorar los resultados que el CarryBot obtiene en cada prueba, es decir, determinar correctamente si el Destino es o no alcanzable, y evitar perder tiempo de forma innecesaria. Esto es importante ya que el consumo de energia, y de tiempo son factores criticos en un sistema embebido.

No obstante, aun si se alcanzara dicha meta, no habria que detenerse. Siempre existe la posibilidad de mejorar el algoritmo planteado, la propuesta en si, asi como tambien de expandirla. Si bien no tenemos certeza de esto, quiza seria una posibilidad la incorporacion de un trabajo conjunto entre el ultrasonido y la propuesta enunciada para lograr resultados aun mas precisos.

Ademas, el Hardware ira evolucionando a lo largo del tiempo, por lo que es muy factible que se den propuestas mas elaboradas que la presente.

6 Referencias

1. Adam Coates, Paul Baumstarck, Quoc Le, and Andrew Y. Ng *. “Scalable learning for object detection with GPU hardware” Internet: https://ai.stanford.edu/~ang/papers/iros09-ScalableLearningObjectDetectionGPU.pdf, 15 de Diciembre de 2009 [Junio de 2018]

2. Joseph Redmon, Santosh Divvala, Ross Girshick, Ali Farhadi *. “You Only Look Once: Unified, Real-Time Object Detection” Internet: https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf, 9 de Mayo de 2016 [Junio de 2018]

3. Adam Coates, Brody Huval, Tao Wang, David J. Wu, Andrew Y. Ng *. “Deep learning with COTS HPC systems” Internet: http://proceedings.mlr.press/v28/coates13.pdf, 2013 [Junio de 2018]