GUION TFG – CONTROL DE SISTEMAS MULTIAGENTE Y SUS APLICACIONES

Buenos días/tardes, me llamo YanRu Wu Jin y voy a presentar mi trabajo de fin de grado titulado “Control de sistemas multiagente y sus aplicaciones”, supervisado por Juan Jiménez Castellanos y Lía García Pérez.

Quiero comenzar discutiendo brevemente el objetivo de este trabajo:

La base de este trabajo será el control de enjambres empleando un algoritmo de control **distribuido**, es decir, un sistema donde cada agente que compone el **enjambre** sea capaz de tomar decisiones por sí mismo, sin tener que depender de un control central, reduciendo así la carga de procesamiento. En este caso nos enfocaremos en un modelo **sin líder**, haciendo del sistema uno más independiente, además de introducir un modelo de control de **transformaciones afines** de la formación deseada. La idea sería ser capaces de implementar este modelo en casos reales de utilidad, como la exploración de una zona complicada sin acceso a GPS, donde los agentes sean capaces de aplicar diversas transformaciones a su formación para adaptarse al entorno.

Una vez aclarado el objetivo pasemos al índice. Exploraremos un poco la notación seguida usando teoría de grafos, las interacciones entre agentes con la Matriz Laplaciana, la dinámica del sistema y los parámetros de movimiento para aplicar las transformaciones afines. Por último, veremos algunas de las simulaciones realizadas durante el trabajo y la conclusión de este.

Teoría de grafos:
Por qué usamos **teoría de grafos**? Esta puede que sea algo que no se entienda bien pero tiene mucho sentido. En este trabajo nos fijamos en los agentes y las conexiones entre ellos, por lo que la teoría de grafos nos provee de un recurso muy útil para explicar estas relaciones. En nuestro caso, los agentes vendrán representados por **nodos**, descritos en un conjunto V y numerados de 1 a n, mientras que la conexión entre ellos (o comunicaciones) vendrá dado por las **aristas**, descritas en otro conjunto de pares de nodos. Ahora, las posiciones de cada agente vendrán apiladas en un solo vector p, que definiremos como configuración. Además, si esta configuración es aquella a la que queremos converger o una transformación de esta la denominaremos como p^*, formación nominal.

Cada agente estará conectado a un número determinado de otros, y la idea será que el sistema no esté completamente conectado. Los agentes a los que está conectado uno serán los **vecinos.** Esto lo podemos ver en este ejemplo (presentar ejemplo).

También vamos a introducir la matriz de incidencia. Esta matriz nos servirá para indicar las conexiones entre agentes. Se define como (mostrar en presentación). Veamos un ejemplo (mostrar ejemplo).

Pasemos ahora a hablar de la matriz laplaciana. En este trabajo, el concepto de matriz laplaciana se refiere realmente a una matriz de estreses, que es una generalización de la laplaciana. Esta matriz representará el peso que tenga cada una de las conexiones del grafo, pudiéndose imaginar como si fueran representativas de unas fuerzas imaginarias que mantienen la formación en la estructura deseada, por ello la tomaremos con pesos, los cual veremos cómo se calculan más adelante. Primero veremos la definición, l_ij, donde cada elemento de la matriz se describirá de la manera que se muestra. Para un mismo nodo (i=j), el elemento será la suma de los pesos de sus vecinos, para un vecino (i\neqj, j\in Vecinos) será -w_ij, y para nodos no relacionados el elemento de la matriz será nulo. Esta descripción la podemos escribir de forma compacta como $L = H$ diag$(\omega)H^T$.

Para obtener los pesos se ha seguido, en la bibliografía, el artículo de Shiyu Zhao. Vamos a repasar un poco cómo se han conseguido. Introduzcamos la notación $\overline{P} = [P(p), \mathbf{1}_n]$, por lo que la formación será la deseada o una transformación afín de esta cuando $\overline{P}L = 0$, ya que las posiciones estarán en el kernel del laplaciano. Por tanto, introduciendo el Laplaciano, $\overline{P}\, H$ diag$(\omega)H^T = \overline{P}\, H$ diag$(\omega)[h_1 \dots h_n] = 0$. Sabiendo que la diag$(\omega)h_i =$ diag$(h_i)\omega$, podemos escribir una matriz E

$$E = \begin{bmatrix} \bar{P}^T(r)H\mathrm{diag}(h_1) \\ \vdots \\ \bar{P}^T(r)H\mathrm{diag}(h_n) \end{bmatrix}$$

Por lo que $E\omega = 0$ y por tanto $\omega \in$ Ker$\{E\}$ los pesos estarán contenidos en el kernel de E. Así, definiendo z_1…z_q como la base del espacio nulo de E (o kernel), podemos escribir los pesos como una combinación lineal de los vectores de esta base.

$$\omega = \sum_{i=1}^{q} c_i z_i$$

Donde ahora lo último que necesitaríamos serían las constantes c_i. Según uno de los teoremas descritos en ese mismo artículo, necesitamos que $U_2^T \mathrm{L} U_2$>0. U_2 será resultante de la descomposición en valores singulares de $\overline{P} = U\Sigma V^T$, donde

U=[U1,U2]. Así reescribimos como $U_2^T H$ diag$(\omega) H^T U_2 > 0$. Introduciendo omega y teniendo en cuenta la propiedad de la diag, definimos una matriz $M_i = U_2^T H$ diag$(z_i) H^T U_2$, por lo que $\sum_{i=1}^{q} c_i M_i \succ 0$, por lo que resolviendo esta inecuación matricial obtendremos las constantes que nos faltaban para averiguar los pesos, y por tanto la matriz completa. Esta inecuación se puede resolver fácilmente en Python, en este caso se ha usado el paquete PICOS para resolverlo.

Todo este cálculo que hemos realizado nos ha otorgado la Laplaciana, que será muy útil para describir la dinámica del sistema, habiendo visto que las posiciones convergen al kernel de L, podremos definir unas entradas de control para conseguir esto. Hemos de tener en cuenta que nuestro sistema tiene una dinámica libre, es decir, no presenta ninguna ligadura y la entrada de control se podrá definir en la dirección que queramos con una magnitud cualquiera. Esta vendrá dada por $\dot{p} = h\bar{L}p = u$, donde la notación barra indica $\bar{A} := A \otimes I_m$ y así $p \to p^* \in$ Ker$\{\bar{L}\}$ convergerán las posiciones hacia la p^* que, como hemos dicho previamente, representan la formación nominal y sus transformaciones afines. Para que se pueda seguir esta dinámica y por tanto estabilizar el sistema, éste debe ser universalmente rígido (además de genérica aunque esto se asume en las posiciones). Para entender la rigidez universal veamos un ejemplo: Si comenzamos en 2 dimensiones, podemos comprobar esta rigidez abstrayéndonos a una dimensión mayor. En esta nueva dimensión si somos capaces de plegar el grafo por una de sus aristas entonces tendremos un grafo no universalmente rígido, como se ve en el ejemplo. Añadiendo una arista más bloqueamos este pliegue, por lo que el nuevo grafo sí es universalmente rígido. Usando la configuración descrita previamente, podemos ver un ejemplo en simulación de esta estabilización. Vemos como, comenzando desde unas posiciones arbitrarias, los agentes se van desplazando hacia una configuración que es claramente una formación que es transformación afín de la nominal.

Estas transformaciones afines que mencionamos las podremos manipular con el uso de unos parámetros de movimiento. Estos los vamos a definir como mu_ij, y vamos a enfocarlo como una simple modificación a la Laplaciana original. De esta manera podremos describir las nuevas entradas de control con una simple sustitución de Laplaciano. Esta modificación vendrá dada, en sus pesos, como un elemento mu_ij multiplicada por una ganancia asociada a la transformación que podremos modificar según nuestros objetivos, y dividiendo por h, para compensar el término de la entrada de control y que las transformaciones solo dependan de una sola ganancia, kappa. Podremos escribir la matriz laplaciana modificada de forma compacta como

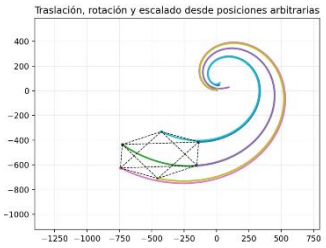
$\tilde{L} = L - \frac{\kappa}{h}MH$, donde los parámetros ahora están contenidos en una nueva matriz M y la de incidencia. Esta matriz M podrá ser separada en cada una de las transformaciones afines posibles, en el caso de 2D serán 6 elementos linealmente independientes: 2 de traslación, 1 de rotación, 1 de escalado y otros 2 de cizallamiento en cada eje. Así la entrada de control es simplemente la misma pero introduciendo el nuevo laplaciano:

$\dot{p} = -h\overline{\tilde{L}}p = -h\overline{L}p - \kappa\overline{MH}p$

Vemos que el control queda bastante simple, donde con la ganancia h controlaremos la velocidad de estabilización y con la kappa (una cada para transformación que queramos aplicar) podremos controlar las transformaciones en el sistema.

Veamos ahora algunas de las simulaciones realizadas a lo largo de este trabajo. Todas estas se han realizado en Python, con la ayuda de paquetes como *numpy* y *matplotlib*, y para animarlas se ha usado una extensión de la última llamada *anímate.* Así podemos observar de manera muy visual cómo el sistema se comporta, desde una configuración donde las posiciones iniciales son las de la formación nominal, bajo la influencia de los diferentes tipos de transformaciones. Los agentes están representados cada uno con un punto coloreado, las trayectorias de cada uno son las líneas de cada color y se han representado las conexiones entre ellos con las aristas punteadas.

Ahora podemos combinar algunas de estas transformaciones con la estabilización, comenzando desde unas posiciones arbitrarias. En este caso se ha aplicado una traslación, rotación y escalado. Es notable indicar que las transformaciones se aplican una sobre la otra, por ejemplo, la traslación + rotación provocan una trayectoria más elipsoidal, y la adición de escalado hace que tenga una forma de caracola.



Este sistema también se puede aplicar en 3D. Se representa de la misma manera a los agentes y a las conexiones, en este caso usando una formación octoédrica. Cabe mencionar que ahora las transformaciones tendrán componentes adicionales: la traslación ahora podrá ser sobre 3 ejes, la rotación también sobre 3 ejes y el cizallamiento igual. El escalado no cambia.

Al igual que con el caso bidimensional, se puede combinar la estabilización con las transformaciones, dándonos el siguiente resultado. Se han empleado traslaciones, rotaciones y escalado.

Como última simulación, se ha comprobado la robustez de este sistema empleando una configuración formada por 18 nodos. Cabe recalcar que en este caso cada agente está conectado sólo con otros 3 vecinos, comprobándose así que no es necesario que

tengan muchas conexiones. El sistema logra estabilizarse correctamente, y el motivo de que cada vez se desplacen más lentamente es debido a que se describe con una exponencial negativa temporal e^-At, por lo que tenemos una estabilidad asintótica y, p->p^* para t-> inf.

Como conclusión, en este trabajo hemos desarrollado correctamente un algoritmo robusto de control distribuido, capaz de guiar a un número significante de agentes hacia una formación nominal y controlarlo mediante una serie de transformaciones afines dadas. Sin embargo, se podría buscar mejorar este sistema:

- Actualmente es incapaz de seguir unas trayectorias predefinidas
- Las posiciones de llegada dependen solo de las posiciones iniciales, por lo que los agentes solo llegarán a la formación nominal original si comienzan desde esta.
- Además en este trabajo se asume una dinámica libre, por lo que sería interesante intentar implementar un sistema que defina las ligaduras de un caso físico y tener en cuenta casos de saturación de velocidades.

Potencialmente, este tipo de control podría llegar a ser usado para usos en exploración (no sé cómo completar esta última sección sin repetirme).