Aplicaciones de sentido común en robots manipuladores

Guillermo Trinidad Barnech

Programa de Posgrado en Informática
Facultad de Ingeniería
Universidad de la República

Montevideo – Uruguay
Febrero de 2023

Aplicaciones de sentido común en robots manipuladores

Guillermo Trinidad Barnech

Tesis de Maestría presentada al Programa de Posgrado en Informática, Facultad de Ingeniería de la Universidad de la República, como parte de los requisitos necesarios para la obtención del título de Magíster en Informática.

Directores:
Ph.D. Prof. Gonzalo Tejera
Ph.D. Prof. Juan Valle Lisboa

Director académico:
Ph.D. Prof. Gonzalo Tejera

Montevideo – Uruguay
Febrero de 2023

Trinidad Barnech, Guillermo

Aplicaciones de sentido común en robots manipuladores / Guillermo Trinidad Barnech. - Montevideo: Universidad de la República, Facultad de Ingeniería, 2023.

XI, 58 p.: il.; 29, 7cm.

Directores:

Gonzalo Tejera

Juan Valle Lisboa

Director académico:

Gonzalo Tejera

Tesis de Maestría – Universidad de la República, Programa en Informática, 2023.

Referencias bibliográficas: p. 49 – 53.

1. Robótica Cognitiva, 2. Física intuitiva, 3. Sentido Común. I. Tejera, Gonzalo, Valle Lisboa, Juan, . II. Universidad de la República, Programa de Posgrado en Informática. III. Título.

INTEGRANTES DEL TRIBUNAL DE DEFENSA DE TESIS

Ph.D. Prof. Bruno Lara Guzmán

Ph.D. Prof. Andrés Pomi

Ph.D. Prof. Álvaro Martín

Montevideo – Uruguay
Febrero de 2023

A mi hermano, mis padres, mis amigos y amigas. Compañeros de camino.

Agradecimientos

Quisiera agradecer en primer lugar a mis tutores, Gonzalo y Juan, quienes me acompañaron durante este proceso de exploración con gran dedicación. También a Pablo Bustos y todo el equipo de RoboLab, quienes me recibieron de la mejor manera y aportaron ideas y comentarios invaluables para este trabajo.

A mi familia y mis amigos, quienes siempre me apoyaron desinteresadamente y aportaron, sin saberlo, una enorme cantidad de información para construir lo que hoy presento.

Por último, quisiera agradecer a la Comisión Académica de Posgrado y al Programa de Desarrollo de las Ciencias Básicas, ya que sin su soporte este trabajo no podría haber sido llevado adelante.

Common Sense is that which
judges the things given to it by
other senses.

Leonardo Da Vinci.

RESUMEN

Las técnicas actuales para el desarrollo de la inteligencia artificial suelen requerir etapas de entrenamiento muy exigentes, con cientos de miles de ejemplos para aprender a resolver las tareas. Debido a esto, ha sido un desafío utilizarlas en robótica autónoma, donde cada intento de resolver la tarea requiere tiempo y normalmente supervisión humana. En los últimos tiempos ha surgido el interés de desarrollar robots con sentido común, capaces de comprender a un mayor nivel de abstracción el mundo a su alrededor y así acelerar su aprendizaje y mejorar su desempeño. La física intuitiva es una parte del sentido común, que refiere a la capacidad de predecir el comportamiento de los objetos en el mundo. Una de las teorías existentes postula que las personas contamos con algo similar a un motor de videojuegos embebido en nuestra mente, que utilizamos para dar sentido a nuestra percepción y predecir posibles eventos a nuestro alrededor. En este trabajo de maestría se lleva esta teoría a la práctica, implementando módulos capaces de convertir la percepción a entidades y luego, de insertarlas en un simulador de físicas y desplegándolos en un robot real. Las ventajas de utilizar esta implementación son demostradas sobre un brazo robótico, en tareas simples de manipulación de cubos. En particular, se presentan experimentos sobre mejora de la percepción, auto-calibración, estimación de posiciones en situaciones de oclusión y una prueba integradora que incorpora algunos mecanismos de sorpresa a una tarea específica de apilado de objetos. En todos los casos la utilización del simulador mejora de manera no trivial el desempeño del robot.

Palabras claves:
Robótica Cognitiva, Física intuitiva, Sentido Común.

Lista de figuras

Tabla de contenidos

Capítulo 1

Introducción

Desde sus inicios, la robótica autónoma buscó suplir a los humanos en labores riesgosas y/o no deseadas. Los primeros avances fueron prometedores, con manipuladores capaces de mover cargas muy pesadas con gran precisión. Esto llevó a un gran desarrollo de la robótica industrial, creando autómatas cada vez más potentes y más flexibles. Así es que hoy en día son robots quienes ensamblan autos, sueldan metales, clasifican fruta e incluso manejan almacenes enteros de productos. El siguiente paso, naturalmente, se dio en dirección a la robótica doméstica, que refiere a la inclusión de robots autónomos en el hogar. Uno de los más claros ejemplos de estas tecnologías son los robots de limpieza, tomados hoy en día como un electrodoméstico más. Sin embargo, las capacidades de estos se encuentran muy alejadas de lo que se concibe normalmente como un robot doméstico. La competencia roboCup@home[45] reúne varios equipos de investigación alrededor del mundo en un desafío para desarrollar robots de servicio. En esta se pueden ver robots de forma humanoide realizando tareas como servir una mesa, cargar un lavavajillas, llenar un carro de compra, entre otras. Sin embargo, el avance en esta área no se asemeja en velocidad a lo que sucedió en la industria. Rápidamente investigadores y desarrolladores identificaron que para que un robot pueda moverse con soltura en nuestros hogares, existe mucho trabajo por delante. La naturaleza desestructurada de los espacios que habitamos complejiza enormemente las tareas. Una tan simple como encontrar un cuchillo en una cocina se vuelve una búsqueda enorme si no se cuenta con las heurísticas correctas para limitarla e incluso una vez localizado, manipular objetos es todo un desafío si el ambiente no es ideal.

Por otra parte y al mismo tiempo, comienza la nueva ola de la inteligencia artificial. Utilizando las técnicas de aprendizaje profundo y aprendizaje por refuerzo, se

desarrollaron agentes capaces de lograr hitos sin precedentes. Es normal hoy en día que existan autómatas capaces de vencer a los humanos en juegos de gran complejidad como el ajedrez [37], Go [36] o Arcade[26]. Este tipo de sistemas logran capturar reglas, interpretar a sus oponentes y desarrollar así estrategias. Estos ejemplos muestran el potencial que tienen estas técnicas como motores de inteligencia, pero en general, sufren de dos grandes problemas. El primero es que siguen desempeñándose en entornos controlados y altamente especializados, es decir, la capacidad de jugar ajedrez es muy impresionante, pero no es transferible a otros problemas. El segundo problema surge cuando queremos utilizar estas técnicas para entrenar robots. Normalmente, el entrenamiento requiere cientos de miles de ejemplos o de interacciones con el problema para lograr resolverlo correctamente. Esto resulta impracticable en la robótica, donde cada interacción requiere tiempo, energía y muchas veces intervención humana.

En respuesta a esta problemática, varias técnicas fueron desarrolladas. El aprendizaje por refuerzo jerárquico [3] es una de estas, donde los agentes construyen habilidades de alto nivel de abstracción que luego pueden utilizar para acelerar su aprendizaje en nuevos ambientes. Existen variados métodos para construir estas habilidades de forma autónoma, como el llamado *Skill Chaining* [17]. Allí los agentes comienzan por construir una política que los lleve al estado final desde estados relativamente cercanos a este. Luego, deberán encontrar nuevas políticas que, comenzando más lejos, lo lleven a un estado donde pueda ejecutar la política previamente aprendida. De esta manera se construyen cadenas de políticas locales (*Skills*) que al ejecutarse secuencialmente logran solucionar el desafío. Esta técnica acelera el aprendizaje de los agentes en ambientes simples, pero requiere algunas simplificaciones al aplicarse en robótica. Por ejemplo, se puede dotar a un robot de habilidades atómicas (empujar, agarrar, girar, etc.) y buscar utilizar *Skill Chaining* para que las combine en búsqueda de soluciones a nuevos problemas [14], pero a medida que se agregan habilidades el aprendizaje se ralentiza, volviendo a requerir muchas interacciones con el entorno.

Interactuar con el entorno real puede ser costoso, pero ¿por qué no utilizar simuladores para realizar el entrenamiento? Creamos una réplica de nuestro robot y hacemos que pruebe una y otra vez la tarea que deseamos solucionar en un entorno simulado, donde podemos acelerar los tiempos y prescindir de la supervisión humana, ya que no hay riesgo de daños. El problema es que una vez terminado este proceso, surge el verdadero desafío: *Sim to real Gap*[6]. Los simuladores, por más potentes que sean, no son el mundo real y es normal que un robot capaz de realizar

la tarea perfectamente en simulación no logre resolverla cuando se enfrente a ella fuera de esta.

Ante esta problemática, donde las técnicas utilizadas en el estado del arte no parecen ser aplicables directamente a nuestra realidad, los investigadores han buscado nuevas direcciones. Históricamente, los desarrollos más exitosos del área han sido inspirados en lo que sabemos sobre sistemas biológicos. El aprendizaje por refuerzo intenta replicar la manera en que los animales aprenden de su propia experiencia [41] y las redes neuronales (utilizadas para el aprendizaje profundo) nacen como imitación del cerebro animal [25]. Es por esto que, una vez más, se busca en nuestro conocimiento de los agentes naturales la respuesta a los problemas actuales.

Las personas tenemos una increíble capacidad para desenvolvernos en el mundo que nos rodea. Es solo cuando uno se da a la tarea de replicar algunas de las habilidades humanas que dimensiona la complejidad de las mismas. Por ejemplo, una tarea tan simple como manipular un objeto que no habíamos visto antes sigue siendo un problema abierto en la robótica autónoma [24]. Y no son solo habilidades motoras: todo el tiempo razonamos sobre los objetos que nos rodean y somos capaces de predecir con sorprendente precisión el comportamiento de los mismos[39]. Sin haber presenciado la situación particular, podemos saber qué pasará si golpeamos la taza de café que se encuentra en la mesa. Podríamos incluso estimar dónde caería el líquido y rápidamente actuar para minimizar el daño.

Esto despertó, en los últimos años, mucho interés. Habilidades cognitivas de este nivel transformarían las capacidades de nuestros autómatas. Se busca entonces dotar a los robots de lo que se llamó *sentido común* [13]. Si bien no existe una definición formal para lo que conocemos como sentido común, este suele separarse en tres grandes categorías:

- **Datos comunes.** Tal vez más alineado con la creencia popular, refiere a los hechos o datos que los adultos promedio consideramos verdaderos sin necesidad de someterlos a debate. Durante mucho tiempo este fue el foco de la investigación en sentido común, donde se construían grandes bases de datos, ontologías y conjuntos de reglas capaces de generar estos conceptos. Hoy en día los avances en modelos de lenguaje, entrenados en grandes volúmenes de datos, suelen capturar estos conceptos y dar respuestas que podrían interpretarse como de sentido común[31], aunque esto sigue bajo debate en la comunidad[48].
- **Psicología intuitiva.** Interactuar con pares es una capacidad que desarrolla-

mos rápidamente, siendo esta una de las principales motivaciones de los bebés. Este continuo deseo de mantener relaciones sociales hace que desarrollemos una extraordinaria capacidad de predicción sobre otras personas. Esta parte del sentido común refiere a cómo comprendemos las motivaciones de los demás, atribuimos intencionalidad y entendemos el comportamiento de otros agentes en el mundo. Esta habilidad es la que nos permite colaborar entre personas muchas veces sin necesitar una comunicación explícita [32].

- **Física intuitiva.** Sabemos que el mundo a nuestro alrededor responde a un conjunto de reglas que hoy en día conocemos. Podemos, en teoría, calcularlas y así poder determinar qué pasará con los objetos que nos rodean. Sin embargo, esto no es lo que las personas hacen para predecir los efectos de sus acciones. Es parte del sentido común comprender los eventos físicos que vemos a diario. Esta habilidad es necesaria para poder desempeñar nuestras tareas y la utilizamos prácticamente todo el tiempo de manera automática, desde una muy temprana edad[40].

Lograr replicar estas habilidades tendría un gran impacto en el desarrollo de la robótica autónoma. Si los robots son capaces de comprender el mundo que los rodea, podrían adaptarse rápidamente a nuevas situaciones, colaborar con humanos de manera natural y generalizar sus habilidades en distintas tareas.

En el marco de este trabajo, nos concentraremos en la física intuitiva, ya que es la que puede tener el impacto más inmediato en las aplicaciones de robótica. Con esta capacidad, los robots serían capaces de razonar en términos de objetos, fuerzas, relaciones y eventos, permitiéndoles predecir cómo se comportará su entorno y utilizar esto como parte de la toma de decisiones. Además, este nivel de abstracción aceleraría los procesos de aprendizaje: es más simple aprender de la experiencia cuando esta se describe en entidades que utilizando directamente las entradas sensoriales.

Sin embargo, los mecanismos detrás de la física intuitiva siguen siendo un misterio. Actualmente, no existe consenso sobre cómo se obtienen o qué procesos lo subyacen [19]. Una de las teorías existentes postula que las personas contamos con algo similar a un motor de videojuegos embebido en nuestra mente, que utilizamos para dar sentido a nuestra percepción y predecir posibles eventos a nuestro alrededor [43].

Esta interpretación de la física intuitiva trabaja a los niveles de abstracción mencionados, transformando la percepción en objetos (y sus propiedades) y colocán-

dolos un mundo simulado. Una de las principales ventajas de esta teoría es que sugiere una implementación computacional directa, y siguiéndola, varios investigadores han logrado construir sistemas que capturan las intuiciones físicas de las personas, realizando predicciones muy similares a estas [4][38].

El objetivo de este trabajo de maestría es llevar la teoría del motor de videojuegos en tu mente a la práctica, implementándola en un robot real. Para esto se desarrollaron distintos módulos sobre una arquitectura de robótica cognitiva, que se encargan de convertir la percepción a entidades y luego, de insertarlas en un simulador de físicas. De esta manera se logra que el robot tenga un mundo virtual donde validar y corregir sus interpretaciones de la entrada sensorial. Las ventajas de utilizar esta implementación son demostradas sobre un brazo robótico, en tareas simples de manipulación de cubos. En particular, se presentan experimentos sobre mejora de la percepción, auto-calibración, estimación de posiciones en situaciones de oclusión y una prueba integradora que incorpora algunos mecanismos de sorpresa a una tarea específica de apilado de objetos.

Capítulo 2

Trabajos relacionados

En este capítulo se resumen algunos de los trabajos más relevantes del área de la física intuitiva. La selección de artículos sigue el desarrollo de la presente investigación, comenzando por perspectivas clásicas con técnicas de aprendizaje para luego volcarse a otras inspiradas en la teoría del motor de videojuegos en tu mente.

Durante los primeros avances, se interpretó a la física intuitiva como la capacidad de predecir el futuro inmediato. Definirla en estos términos tiene la ventaja de convertirlo en un problema atacable con los métodos tradicionales. Si el objetivo es saber qué pasará, basta con entrenar un modelo que dado un estado, nos diga el siguiente. Para esto se puede utilizar aprendizaje supervisado, entregando (por ejemplo) un video y entrenando el sistema para generar la siguiente imagen [30]. Orientando esta idea más hacia la física, podemos encontrar redes neuronales entrenadas para predecir: estabilidad en torres de bloques [20] o los efectos tras aplicar una fuerza [28]. En muchos casos, estos sistemas son capaces de realizar la predicción, pero ¿significa esto que logran capturar las dinámicas del mundo?. En su trabajo, Ullman et al.[43] argumentan que no es el caso, ya que solamente razonan en términos de píxeles y no de objetos, fuerzas y masas. Postulan que esto limita enormemente la capacidad de generalizar los conocimientos que capturan, apelando a cómo fallan cuando se las presenta con nuevos objetos o condiciones ligeramente distintas a su conjunto de entrenamiento (como puede ser el uso de más bloques en una torre).

Contrapuesta a estas posturas, existe otra visión que consiste en el uso de simuladores como “modelos ejecutables” donde se pueden reproducir las escenas y extraer conclusiones basadas en sus desenlaces. En esta escuela podemos encon-

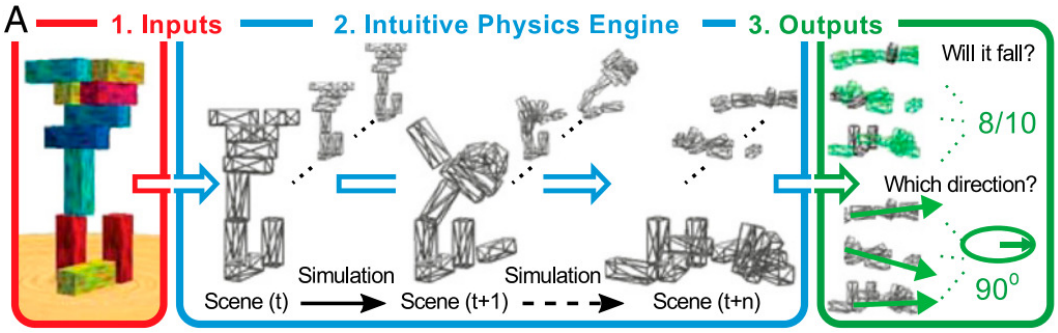


Figura 2.1: Predicción de estabilidad en torres de bloques utilizando simulación. Tomado de [4]

trar el trabajo de Battaglia et al. [4], donde utilizan un motor de videojuegos para simular las intuiciones físicas de las personas. En sus experimentos, presentan una torre de bloques y es la tarea del sistema predecir si esta caerá y en ese caso, en qué dirección. Su método consiste en generar varias instancias de la escena a partir de la distribución de probabilidad del observador sobre sus percepciones, ejecutar la simulación y luego unificar los resultados para dar una distribución de posibles desenlaces (figura 2.1).

Como forma de validar este proceso, se correlacionaron las respuestas que el sistema daba con las de participantes humanos. Esto mostró una gran coincidencia entre las respuestas, reforzando la noción que es este el mecanismo que los humanos usamos como motor de física intuitiva. Para intentar explicar esta coincidencia, los autores presentan más adelante la teoría del motor de videojuegos en tu mente [43]. Allí ahondan en algunos de los conceptos detrás de los motores de videojuegos y cómo estos se parecen a los descubrimientos sobre la mente. Uno de los argumentos que presentan es que el objetivo de los motores de videojuegos es simplemente aparentar realismo, no ser fieles a las físicas. Y es justamente esto lo que los convierte en un gran candidato para las simulaciones, ya que el resultado de esto es construir sistemas condicionados a nuestras propias expectativas del mundo físico. Nuestras predicciones sobre el comportamiento de los objetos que nos rodean han sido estudiadas y presentadas como parte de los llamados conocimientos nucleares[16]. En un intento por capturar estas expectativas e inspirados en la teoría del motor de videojuegos, Smith et al. [38] desarrollaron un modelo que combina las predicciones de un motor físico con técnicas de remuestreo, para hacer seguimiento de objetos en video. Luego, para evaluar si logra capturar los conceptos básicos sobre objetos, se le presentan situaciones físicamente imposibles (figura 2.2), registrando las diferencias entre las predicciones del modelo y la realidad como una medida de sorpresa.

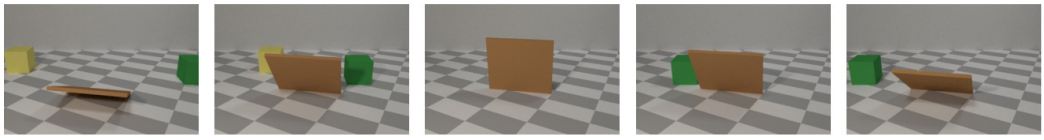


Figura 2.2: Ejemplo de video físicamente imposible, donde el cubo amarillo desaparece detrás del oclusor. Tomado de [38]

A su vez, las mismas situaciones son presentados a voluntarios, que deben indicar una medida de sorpresa. La experimentación demostró una gran similitud entre las respuestas de las personas y la medida de sorpresa del modelo, fortaleciendo una vez más este mecanismo de física intuitiva.

Los desarrollos y avances presentados hasta el momento son de naturaleza experimental, sin ser implementados en un *pipeline* de percepción real ni embebidos en una tarea robótica. Sin embargo, muchos grupos de investigación han tomado inspiración en estas ideas y llevado adelante proyectos para mejorar el desempeño de sus robots utilizándolas.

Un ejemplo de implementación es el presentado por Mania et al. [21], donde utilizan Unreal Engine 4 como simulador mental de un robot doméstico autónomo. La forma de utilizar esta simulación difiere un poco de la idea original, siendo que los autores se centran en las capacidades de foto realismo de su plataforma. En su propuesta, el agente contiene los modelos geométricos exactos de todos los objetos junto a las texturas necesarias para representarlos de forma realista. Así, el robot inyecta en la simulación cada una de las detecciones que realiza, generando una réplica virtual del mundo que percibe. Gracias a las físicas del sistema, estas percepciones sufren algunas modificaciones en su posición, como puede ser evitar que floten sobre una superficie. Una vez sucedido esto, renderizan en el mundo virtual lo que debería ver la cámara del agente y la comparan con el *stream* real. Con esta información se pueden corregir las poses e incluso descartar falsas detecciones de objetos.

Otra inmediata ventaja de contar con un simulador embebido es la posibilidad de probar diferentes situaciones en búsqueda de una que cumpla las condiciones deseadas. Es de esta forma que lo utilizan Mösenlechner et al. [27] combinando esta capacidad con proyección simbólica abstracta. Una vez que cuentan con un plan en alto nivel, los autores utilizan el lenguaje de programación lógica Prolog para encontrar posibles soluciones reales de estos estados abstractos, como puede ser encontrar un lugar donde colocar un plato de modo que este sea estable, visible

y alcanzable por el robot. Es justamente en la resolución de estos predicados que utilizan el motor de físicas *Bullet*[10] y OpenGL. Para el ejemplo planteado, se generan posiciones candidatas al azar y se reproducen en el mundo simulado, una vez allí pueden utilizar OpenGL para renderizar una imagen y detectar si el objeto es visible por el robot o simular unos segundos y determinar si es una posición estable.

Por último, se resalta el trabajo desarrollado por Sallami et al. [35] sobre razonamiento geométrico basado en simulación. Siguiendo las ideas sobre física intuitiva, los autores presentan algunas ventajas de contar con un módulo de razonamiento físico y lo implementan como un simulador que se encuentra sincronizado con la percepción del robot. En sus experimentos presentan al sistema diversas cajas y cubos que este deberá reconocer y seguir en tiempo real. Cuando los detecta, inserta sus representaciones en el motor de físicas *Bullet* y actualiza su posición en concordancia con estos. Gracias a esta representación, el robot es capaz de razonar geométricamente sobre objetos que se encuentran completamente ocultos y detectar relaciones como *dentro de* o *sobre*. Presentan un conjunto de experimentos de gran interés donde se ve la potencia de esta técnica, logrando mantener una representación exacta de la realidad a pesar de no percibir algunos de los objetos que la componen.

Durante los siguientes capítulos se detalla el sistema desarrollado en este trabajo, el cual toma muchas de las ideas aquí presentadas. Siguiendo el trabajo de Sallami et al.[35] se implementan diversos mecanismos para sincronizar la percepción del robot con un simulador determinístico y se llevan adelante pruebas de seguimiento de objetos durante oclusiones. La simulación permite obtener las correcciones en posición reportadas por Mania et al.[21], pero no se utiliza un entorno foto realista para detectar discrepancias, sino una medida de distancia entre simulación y percepción. Esta medida es interpretada como una señal de sorpresa, inspirada en los conceptos presentados por Smith et al.[38] y utilizada para reconocer situaciones anómalas. Cuando el sistema percibe una señal de sorpresa alta, busca darle explicaciones con un método inspirado en el trabajo de Mösenlechner et al.[27], tomando información simbólica de alto nivel y probando en el simulador diferentes escenarios, buscando uno que explique su percepción.

Capítulo 3

Métodos desarrollados

Esta sección describe las decisiones de diseño, su inspiración y los detalles de la implementación de cada parte del sistema. Muchos de los aportes presentados en los siguientes capítulos fueron previamente publicados en un artículo aceptado en la conferencia ROBOT2022 [42].

3.1. Entorno de trabajo

Siendo el objetivo de este proyecto investigar las ventajas asociadas a un simulador embebido como motor de sentido común físico, se decidió utilizar como caso de estudio un robot manipulador. En particular, se utiliza una configuración *Table Top*, donde un brazo robótico debe mover objetos que se encuentran en una mesa, a la cual este está fijado. En este tipo de tareas es de vital importancia la precisión que el sistema tenga a la hora de detectar y ubicar objetos, ya que un error de milímetros puede afectar las consecuencias de una acción por completo y dejar el sistema en un estado inconsistente o erróneo.

La posición y dimensiones de la mesa, al igual que la posición de la base del brazo, son configuradas inicialmente y conocidas por el sistema.

3.1.1. Arquitectura

Una parte fundamental para cualquier desarrollo significativo en robótica autónoma es la elección de un entorno validado de trabajo. Esta selección deberá tomar en cuenta los objetivos del trabajo para alinearlos lo más posible con el entorno que se vaya a utilizar y acompañar de la mejor manera la implementación. En este proyecto se busca explorar los beneficios de extender las capacidades cognitivas de un

robot manipulador, por lo que se decidió utilizar una arquitectura de robótica cognitiva. Estas difieren de las arquitecturas convencionales en su origen e inspiración. Nacen de desarrollos en ciencias cognitivas y se construyen replicando estructuras y mecanismos presentes en los animales y seres humanos, normalmente buscando la construcción de sistemas capaces de adaptarse rápidamente a su entorno.

Kostseruba y Tsotsos realizaron una exhaustiva revisión de las arquitecturas cognitivas existentes [18] y destacan solo unas pocas que tienen la capacidad de implementar las distintas habilidades necesarias para resolver escenarios complejos. Una de estas es la arquitectura CORTEX, desarrollada en la Universidad de Extremadura, España [7], cuya principal característica es la existencia de una memoria de trabajo unificada, donde se pueden representar tanto datos sensoriales como símbolos de alto nivel de abstracción. La memoria de trabajo es un concepto complejo de las ciencias cognitivas, que en términos generales se refiere a todos los sistemas que permiten mantener información disponible mientras se realiza una tarea. El modelo más citado es el propuesto por Alan Baddeley et al.[1] hace más de 40 años y sus refinamientos posteriores. El presentar detalle de esa y otras arquitecturas y sus bases empíricas excede a este trabajo, pero puede encontrarse en una revisión más cercana en el tiempo[2].

En términos generales, la arquitectura CORTEX es un conjunto de agentes cooperando para lograr un objetivo. Estos son independientes y paralelos y deben ser diseñados de forma que cada uno cumpla una función básica. Los agentes no conocen de la existencia de los demás y no pueden comunicarse explícitamente entre ellos. La única conexión entre los distintos módulos es la memoria de trabajo, donde cada uno podrá insertar la información que produzca y consultar la producida por los demás. Se puede interpretar a los agentes como los diferentes módulos del cerebro, que cumplen su tarea y ofrecen sus resultados en un espacio común. Para desarrollos inspirados en ciencias cognitivas esta característica es fundamental, ya que permite implementar por separado cada proceso cognitivo y que estos generen en la memoria de trabajo un estado coordinado. Al ser este compartido, pueden detectarse rápidamente discrepancias y disparar mecanismos de resolución de conflictos de alto nivel, simulando también la mente humana [23].

La implementación de los agentes de CORTEX está basada en el *framework* RoboComp [22].

Figura 3.1: Organización general de la arquitectura CORTEX. Varios agentes independientes se comunican entre sí a través de una memoria de trabajo compartida, donde representan el estado actual de su entorno.

3.1.1.1. Memoria de trabajo

Lo que diferencia esta arquitectura de muchas otras es el concepto de una memoria de trabajo dinámica y compartida entre todos los participantes de la solución. En la figura 3.1 se presenta la organización general de esta arquitectura, compuesta por agentes independientes que aportan a la construcción de una representación unificada de su entorno y estado.

La memoria de trabajo es llamada *Deep State Representation* (DSR) por su naturaleza híbrida, fusionando información semántica y geométrica. Conceptualmente, representa una red de entidades conectadas por las distintas relaciones entre ellas. Formalmente, se representa con un grafo dirigido donde tanto los nodos como las

aristas contienen atributos propios. Los nodos representan conceptos como objetos, planes, robots, etc. y las aristas los relacionan tanto conceptualmente como geométricamente. Dos nodos pueden tener muchas relaciones entre ellos, pero solo una puede ser geométrica. Las aristas llamadas RT (del inglés *Rotation-Translation*) representan la posición de un nodo con respecto a otro y sus atributos son los valores de translación (x, y, z) y rotación $(roll, pitch, yaw)$ entre ellos.

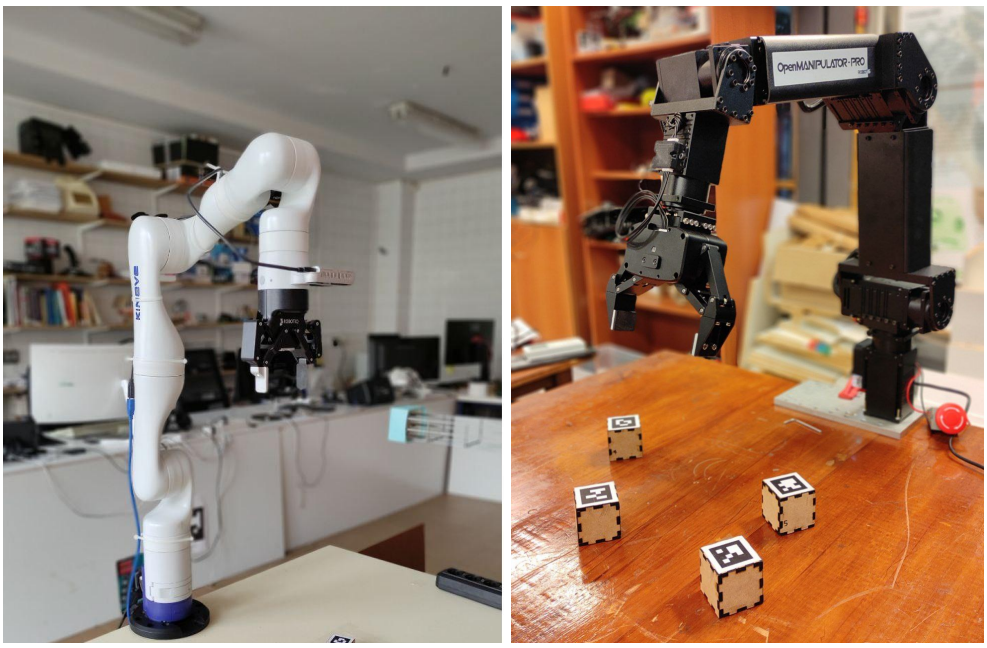
Utilizar una representación unificada del entorno puede traer consigo problemas de *performance* en el acceso a los datos, generando un cuello de botella para la arquitectura. Para solucionar esto mientras se minimiza también el tráfico de información, se utiliza un *middleware* de alto rendimiento que implementa RTPS (del inglés *Real-time Publish-Subscribe Protocol*) [11] configurado para utilizar *Reliable Multicast* sobre UDP. Con esto, la información generada por cualquier agente es enviada en *broadcast* a todos los demás. A su vez se utilizan los tipos de datos *Conflict-free Replicated Data Types* (CRDT) que presentan estructuras seguras de modificar de forma asíncrona y distribuida, sin un servidor central.

Juntando estas decisiones de diseño, cada agente es capaz de editar su propia copia local del DSR de manera asíncrona. Estos cambios locales son enviados a través de la red como modificaciones incrementales que al ser recibidas por los demás son unificadas. De esta forma, la memoria de trabajo solo existe como el conjunto de copias locales de los agentes, las cuales convergen a un mismo estado cuando las ediciones se detienen.

3.1.1.2. APIs disponibles

Para la utilización de CORTEX se disponen de varias APIs para Python y C++. Dos de las más importantes para las implementaciones a presentar en este trabajo son la API de acceso al grafo y la API *Inner Eigen*. La primera permite a los agentes tanto la lectura como la modificación de la memoria de trabajo. Utilizándola, los agentes pueden acceder a nodos y aristas del grafo en cualquier momento. Pone a disposición las funciones básicas de inserción y modificación, que una vez ejecutadas se propagan a los demás agentes de forma automática.

Dentro del grafo se encuentra información espacial, presentada como un árbol de aristas RT entre los nodos. La API *Inner Eigen* nos permite realizar operaciones sobre este, como obtener la posición de un objeto desde un marco de referencia, o relaciones espaciales entre objetos no directamente relacionados.



(a) (b)

Figura 3.2: Brazos robóticos utilizados. (a) Kinova Gen 3 equipado con la pinza LF-85 de Robotiq. (b) Open Manipulator Pro equipado con la pinza RH-P12-RN.

3.1.2. Robots

Durante la experimentación se utilizaron dos brazos robóticos distintos de *Kinova* y *Robotis*, en entornos de trabajo independientes.

El manipulador *Kinova Gen 3* equipado con una pinza *Robotiq 2F-85* (Figura 3.2 (a)) fue utilizado para las pruebas 4.1 y 4.2, llevadas adelante durante una pasantía en el laboratorio RoboLab, de la Universidad de Extremadura. A este se le agregó una cámara estereoscópica *RealSense D415* fijada a su muñeca como única capacidad perceptiva.

Para las demás pruebas, llevadas adelante en el laboratorio de Robótica de la Facultad de Ingeniería de la Universidad de la República, se utilizó el *Open Manipulator Pro* desarrollado por Robotis equipado con la pinza *RH-P12-RN* (Figura 3.2 (b)). En este caso no cuenta con una cámara fijada a su estructura. Las capacidades perceptivas vendrán también de una cámara estereoscópica *RealSense*, pero en este caso es el modelo D455 y se encuentra colocada en un marco sobre de la zona de trabajo.

Ambos modelos son ampliamente utilizados en entornos académicos y de desarrollo, siendo un estándar para este tipo de investigaciones.

3.1.3. Objetos

Los objetos a utilizar en tareas de manipulación definen en gran medida la dificultad y foco de la misma. En este trabajo simplificamos las formas, siendo los objetos cubos de cuatro centímetros de lado, ya que no es el foco desarrollar técnicas de agarre y esta geometría permite la manipulación sin mayores inconvenientes.

Para la detección de los cubos, se les coloca una marca fiducial AprilTag [29]. Estas permiten detectar la posición y orientación de los cubos de manera simple y rápida. A pesar de estas simplificaciones, debido al pequeño tamaño de las marcas y la resolución de las cámaras, el sistema perceptivo sufre de errores significativos a la hora de estimar las posiciones.

Los desarrollos presentados en este trabajo son independientes de este *pipeline* y utilizan las interfaces estándar de CORTEX, por lo que podría reemplazarse esta configuración por una más genérica sin afectar el resto de módulos.

3.1.4. Simulador

Como parte central de esta propuesta, el robot cuenta con un simulador como parte de su memoria de trabajo. Al utilizar la arquitectura cognitiva CORTEX, se decidió combinarla con el simulador Coppelia Sim [34] (ex V-REP), ya que existían varios proyectos anteriores combinando estas tecnologías. Para conectarlo con nuestro caso de uso particular se utilizó la API ZeroMQ [9] para invocar scripts de control implementados en el lenguaje LUA encargados de interactuar directamente con la simulación.

En la bibliografía de física intuitiva, se suele sugerir simuladores no determinísticos como motor para estas inferencias. Esto se debe a que normalmente se utiliza el sentido común para predecir, teniendo entonces que tomar en cuenta las incertidumbres naturales del entorno. Coppelia es un simulador determinístico, pero al utilizarlo solamente como apoyo para corregir la percepción no es necesario realizar proyecciones temporales, por lo que no es un problema.

3.2. Agentes

En CORTEX, los procesos son llamados agentes. Estos ejecutan paralelamente y a una frecuencia estipulada previamente. No existe comunicación explícita entre ellos, pero todos tiene acceso al DSR, la memoria de trabajo compartida. A con-

tinuación se presentan algunos de los agentes desarrollados y sus funcionalidades. Todos han sido implementados en Python 3, aprovechando algunas bibliotecas que este lenguaje pone a disposición.

3.2.1. Controlador del brazo

Uno de los primeros agentes implementados es el encargado de mover el brazo y la pinza a las posiciones deseadas. En el DSR, existe un nodo llamado *Gripper* donde se representan varios de los atributos necesarios para que cualquier agente actúe sobre el manipulador. En particular, este posee un atributo *Target Position*: un vector de seis componentes indicando la posición y orientación a la que deseamos el brazo se dirija. Cuando cualquier agente modifica este atributo, el agente controlador es el encargado de enviar los comandos necesarios para que el manipulador llegue a dicha posición. De igual manera, se tiene un atributo *Target Finger Distance* para el control de la apertura de la pinza.

Al mismo tiempo, es este agente el encargado de actualizar la posición real del robot. Esto lo realiza cambiando los valores de la transformación espacial (RT) hacia este a partir de la información proporcionada por los encoders del brazo.

Por la naturaleza paralela y distribuida de este control, el nodo *Gripper* contiene también un atributo booleano *Robot Occupied*, que indica que el brazo se encuentra en movimiento. Si a pesar de estar ocupado un agente modifica la posición destino, se detiene la acción y se dirige al nuevo punto.

Se desarrollan dos versiones de este agente. Para controlar el brazo Kinova Gen3 se utiliza la biblioteca Kortex [15], desarrollada por Kinova Robotics. En el caso del Open Manipulator Pro, se utilizó el paquete de ROS MoveIt [8] junto con un nodo propio desarrollado para recibir las posiciones destino generadas en Cortex y enviarlas al brazo.

Este agente intenta ejecutar a la máxima velocidad posible.

3.2.2. Controlador de cámara

Este agente se encarga de la comunicación con las cámaras del sistema. Utilizando las bibliotecas para Python de RealSense [33] obtiene toda la información necesaria y publica como atributos del nodo *Camera* tanto las imágenes RGB y de profundidad como parámetros importantes de las mismas como pueden ser los puntos focales.

Este agente intenta ejecutar a la máxima velocidad posible, limitado por la frecuencia de muestreo de las cámaras (30 fotogramas por segundo).

3.2.3. Detector de cubos

Como se mencionó anteriormente, cada uno de los cubos cuenta con una etiqueta AprilTag única pegada a una de sus caras. La tarea de este agente es tomar la información del nodo cámara e insertar los cubos que detecte en la memoria de trabajo.

Utilizando la imagen RGB y la biblioteca AprilTag [5], determina la región donde se detecta la etiqueta y su orientación. Para estimar la distancia se consulta la imagen de profundidad. Con esta información, el agente inserta en el DSR un nodo representando el cubo detectado y agrega una arista espacial (RT) desde la cámara hacia el mismo, la cual actualizará siempre que siga detectándolo.

Las detecciones suelen ser ruidosas y proclives a falsos positivos. Es por esto que se implementa un suavizado exponencial (ecuación 3.1) para actualizar las poses. Para minimizar la actividad en el grafo, solo se publican las nuevas posiciones si estas superan un umbral mínimo de diferencia con la actual. Los valores para estos parámetros se pueden ver en el anexo 1.1.1.

$$current\_position = \alpha \times detected\_position + (1 - \alpha) \times current\_position \quad (3.1)$$

Este agente ejecuta a un máximo fijado de 20 Hz.

3.2.4. Controlador de la simulación

La parte central de este trabajo se puede ver en la labor de este agente. Las secciones anteriores representan los agentes encargados de las tareas perceptivas y motoras, clásicos en despliegues de robótica autónoma.

El rol principal que debe cumplir es sincronizar (lo mejor que pueda) el modelo del mundo que está en la memoria de trabajo con el simulador. Esto involucra una comunicación bidireccional. En una primera instancia, el agente toma los cubos presentes en el DSR y, respetando la posición donde son detectados, los inserta en el mundo simulado.

A su vez, se encarga de insertar en la memoria de trabajo las aristas de tipo *Virtual_RT*. Estas contienen la misma información que las aristas espaciales, pero

representan la posición de los cubos dentro de la simulación. Esto servirá como una sugerencia desde el motor de sentido común, una creencia sobre la verdadera pose de los objetos. un diagrama de secuencia describiendo el proceso completo es presentado en la figura 3.3 Veremos más adelante cómo esta simple funcionalidad es capaz de aportar una gran cantidad de valor a nuestro robot.

Por último, se encarga de impactar los movimientos del brazo en el mundo simulado. Esto lo hace como respuesta a cambios en el atributo *Target Position*, fijando este objetivo en el motor de cinemática inversa del simulador, quien se encarga de calcular las posiciones a de cada articulación y ejecutar el movimiento.

En la figura 3.4 se puede observar un estado del DSR generado por los agentes presentados anteriormente. Se incluyen en la figura los agentes y flechas indicando de qué nodo o arista obtienen información y qué hacen con esta. Por ejemplo, podemos ver que el controlador de la simulación obtiene la información de las aristas RT hacia los cubos y la utiliza para insertarlos en la simulación. Este diagrama no es completo, ya que para algunas de las operaciones los agentes requieren información global. En el ejemplo que dimos, el controlador de la simulación debe cambiar el marco de referencia de las aristas que toma, utilizando toda la cadena de transformaciones espaciales desde el nodo mundo a cada cubo.

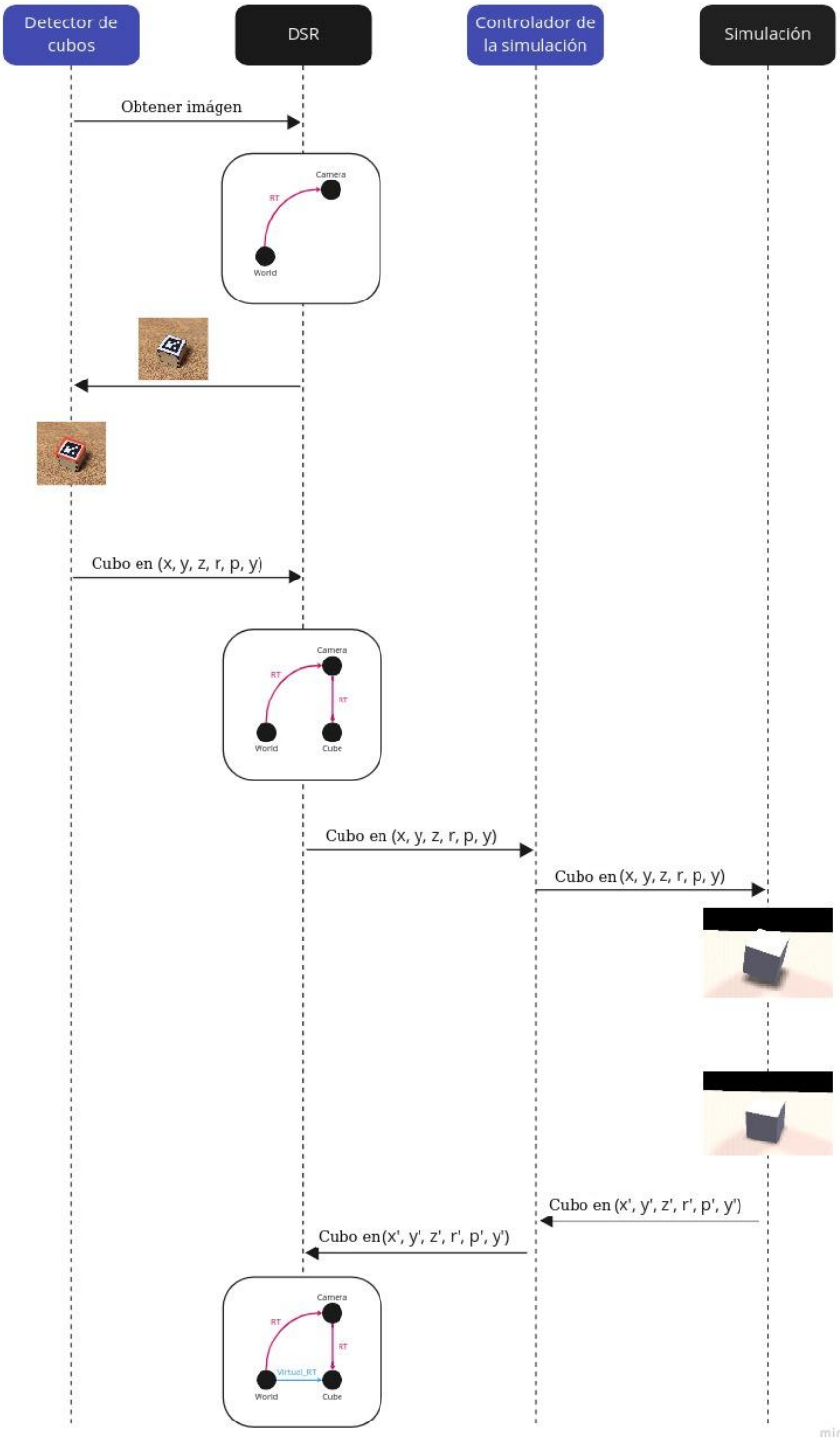


Figura 3.3: Diagrama de secuencia para sincronizado de posiciones de cubos. En primer lugar, el agente detector de cubos, tomando la información de la cámara, detecta e inserta en el DSR un cubo y su pose. Esto genera que el agente controlador de la simulación inserte en el simulador un cubo en la pose reportada. Al aplicar las físicas, la simulación genera una nueva pose físicamente posible para el cubo y el controlador inserta esta información en el DSR como una arista *Virtual_RT*

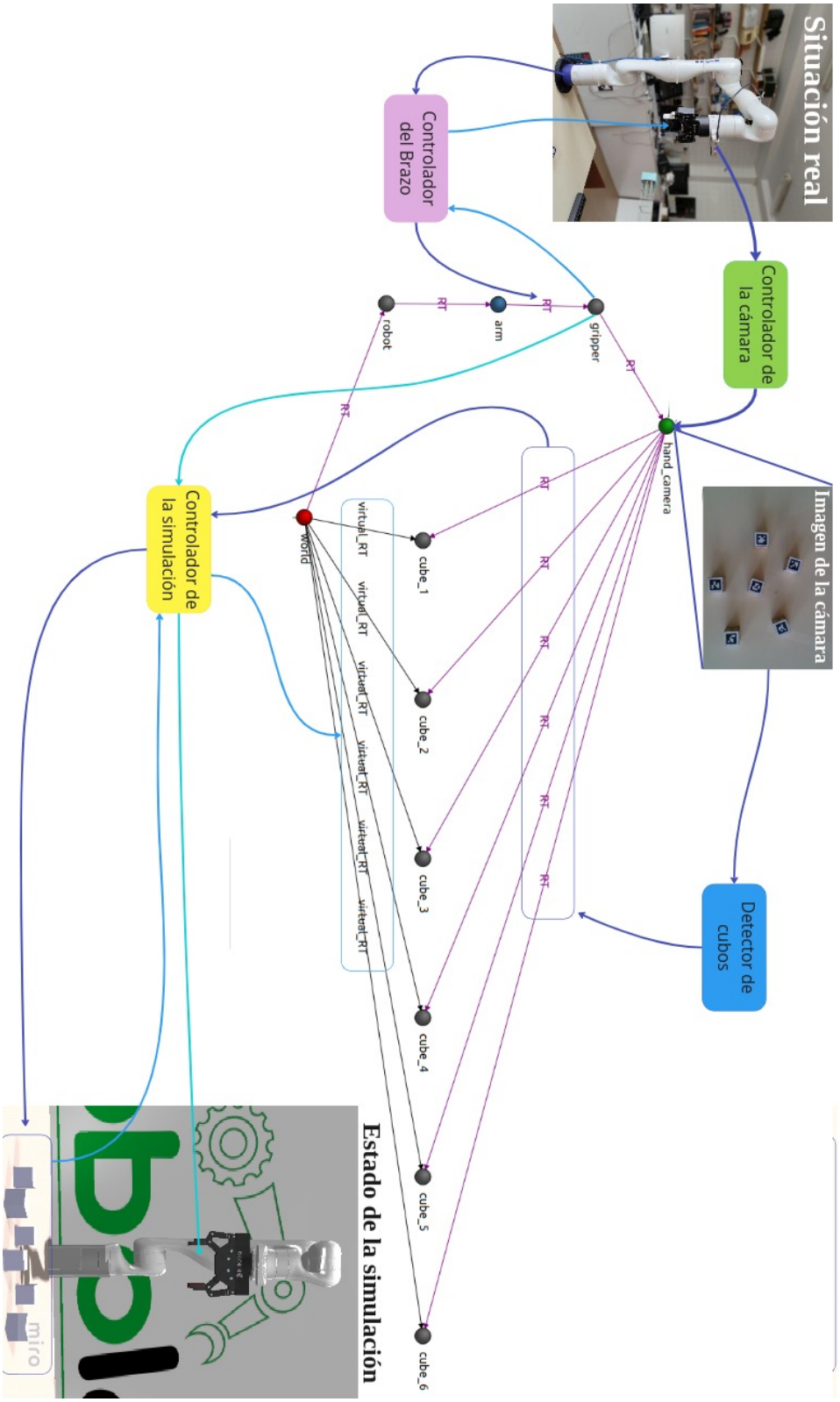


Figura 3.4: Diagrama de la interacción entre cada agente y el DSR. Los colores de las flechas indican la correspondencia entre entradas y salidas para un mismo agente.

Capítulo 4

Experimentos en robots

En este capítulo se presentan las principales pruebas realizadas al sistema presentado. En ellas se ilustran algunos de los beneficios que se pueden obtener al embeber un simulador en la arquitectura del robot. Los resultados obtenidos no tienen como objetivo probar definitivamente la implementación de nuestra propuesta y, por lo tanto, no son estadísticamente exhaustivos. El objetivo es demostrar la gran cantidad de oportunidades que emergen al utilizar esta configuración, sin requerir una gran cantidad de desarrollo extra. En cada sección se presentan los nuevos agentes implementados (si los hubiese) y las pruebas realizadas.

Por claridad en la lectura, no se mencionan los valores de algunos parámetros de configuración de las pruebas. Estos se presentan en el anexo 1.1.

4.1. Percepción basada en modelos

La detección y localización de objetos suele ser un proceso ruidoso y propenso a errores. Incluso utilizando un *pipeline* tan simplificado como el nuestro donde cada objeto tiene una marca única y asimétrica, problemas de iluminación, la distancia de la cámara al objeto y la resolución utilizada pueden generar errores que luego impacten en la *performance* del sistema.

Cuando las personas observamos objetos sobre una mesa, asumimos que es esta que les está dando soporte y, por lo tanto, estarán en contacto. Este razonamiento no suele ser consciente, sino que podría atribuirse a nuestro sentido común físico: “los objetos no pueden flotar en el aire”.

En nuestra configuración, al utilizar varios cubos, suele darse la situación en que el sistema percibe que estos se encuentran en una posición que sabemos que es

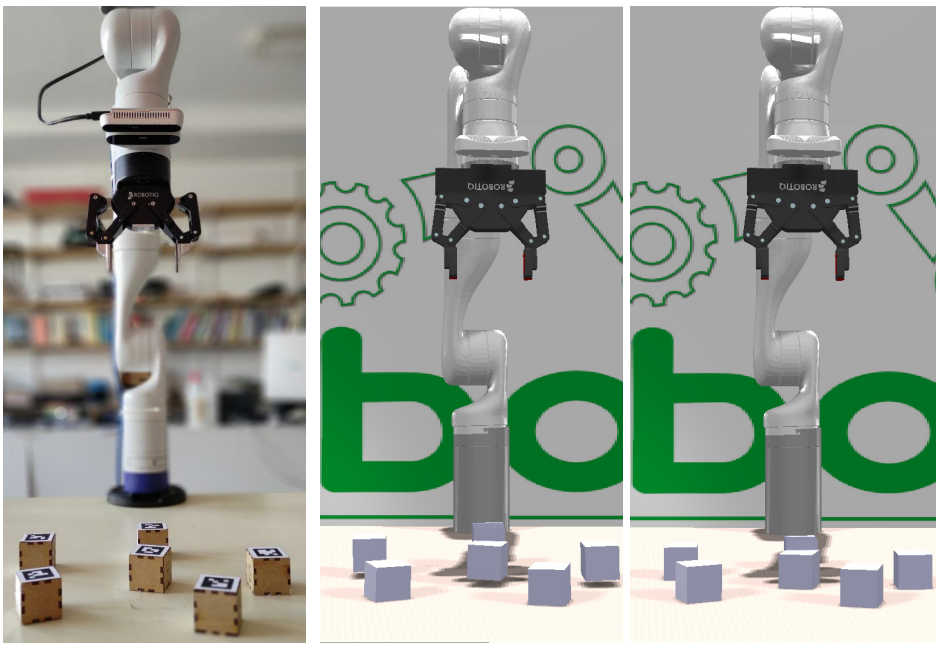
físicamente imposible (asumiendo que no existe intervención de fuerzas o agentes que no vemos). Los casos más comunes son objetos que se encuentran flotando o incrustados dentro de otros, pero errores en la orientación también pueden generar estos problemas. Combinando la simulación y los modelos geométricos de los objetos, el agente controlador de la simulación es capaz de corregir las detecciones de manera automática e inmediata.

En la figura 4.1 se puede observar este comportamiento. Primero el sistema se enfrenta a una configuración particular de cubos (figura 4.1(a)) y debido a los problemas comentados, resulta en la estimación de la escena presentada en la figura 4.1(b) donde se puede observar que varios de los cubos estarían flotando. Luego, gracias a las físicas del simulador, la fuerza de gravedad corrige estas posiciones, como se puede ver en la figura 4.1(c). Esta nueva información se publica en la memoria de trabajo, logrando que cualquier agente tenga acceso a una percepción corregida por un proceso que obedece las leyes de la física. A pesar de no tener garantía que estas correcciones sean correctas, es altamente probable que sea el caso, ya que simplemente hemos llevado nuestra percepción a la explicación físicamente posible más parecida. Ejemplos de esta funcionalidad ya han sido demostrados con anterioridad [35] [27]. En las siguientes secciones veremos cómo estas correcciones tienen ventajas más allá de la ya importante mejora de la precisión.

4.2. Auto calibración

Como resultado de la percepción basada en modelo descrita anteriormente, el sistema contiene para cada cubo un error residual. Es decir, la diferencia entre lo reportado por el sistema perceptivo (registrado en las aristas RT hacia el cubo) y las correcciones proporcionadas por el simulador (aristas Virtual_RT). Esto puede responder a un error en la estimación de la pose, pero también puede y suele ser causado por una mala calibración del robot. Para estimar la posición absoluta de los cubos, debemos utilizar la posición del brazo e incluso la de la cámara con respecto a este. Estas transformaciones no son libres de error y pueden ser la causa de las discrepancias. En particular, las transformaciones estáticas hacia los sensores (en nuestro caso del brazo a la cámara) son medidas por humanos y colocadas manualmente, siendo de las principales causas de estos errores sistemáticos. Para corregirlo, exploramos la posibilidad de seguir las derivadas dadas por este error residual, en búsqueda de una mejor calibración para esta transformación.

Definamos c_i^r como la posición real del cubo i, p^r como la pose de la cámara



(a) (b) (c)

Figura 4.1: Corrección de detecciones utilizando el simulador. Cuando el robot se encuentra el estado (a) utilizando la información de la cámara RGBD, reporta la estimación de la escena presentada en (b). Esta sugiere que varios de los cubos se encuentran flotando sobre la mesa. Luego de aplicar las físicas del simulador, se genera la escena (c), donde las posiciones han sido corregidas.

con respecto al marco de referencia de su padre (nodo inmediatamente anterior en la cadena cinemática) y c_i^e como la pose del cubo i estimada por el agente perceptivo. Usando C^e (el conjunto de todos los c_i^e) el simulador es sincronizado, y luego que se aplican las físicas de este, se obtiene c_i^s , como la pose corregida para el cubo i.

En el siguiente experimento, buscamos encontrar un valor para p^r que minimice la distancia entre C^e y C^s, utilizando la estimación del simulador como aproximación de C^r. Para calcular esta diferencia se utiliza el promedio de las distancias entre las dos estimaciones, calculadas utilizando la fórmula 4.1. Una vez obtenida esta medida de error, se dispara un proceso de minimización utilizando el método de *Powell* implementado en la biblioteca *Scipy* [44].

$$dist(c_i, c_j) = K \times (1 - \langle c_i^{rot}, c_j^{rot} \rangle^2) + ||c_i^{trans} - c_j^{trans}|| \tag{4.1}$$

Para realizar las pruebas asociadas a esta funcionalidad, se implementa la primer versión de un nuevo agente que llamaremos *Razonador Físico*. Para este ejercicio en particular, su labor consiste en monitorear las discrepancias entre percepción (C^e) y corrección (C^s) y en caso que esta supere un umbral predefinido, disparar el proceso de minimización en búsqueda de una mejor calibración.

Una recalibración puede ser necesaria por variados motivos. Algunos de los más interesantes siendo (I) el sistema se encontraba bien calibrado, pero sufrió un cambio inesperado (por ejemplo un pequeño golpe que cambie la posición de la cámara) o (II) la calibración inicial del sistema era errónea y se debe encontrar la correcta.

Siguiendo el caso I, imaginemos que el robot ya detectó algunos cubos, los colocó en simulación y obtuvo sus poses corregidas. Esto de por sí no debería ser suficiente para disparar al agente de recalibración. Pero en algún punto de su tarea el sistema vuelve a percibir estos cubos, esta vez en una configuración lo suficientemente distinta a lo reportado por el simulador. Muchos motivos pueden explicar esto, pero el sistema dispara el proceso de recalibración de cualquier modo, en búsqueda de una p^r que logre eliminar este error. La función a minimizar toma como entrada una posible transformación espacial entre el brazo y la cámara. Con esta y las transformaciones espaciales reportadas desde la cámara a los cubos y desde el mundo al brazo, calcula dónde estarían los cubos en el marco de referencia global. Esto se toma como el nuevo C^e y se define la función de error (calculada con el algoritmo 1) a ser minimizada por el agente.

Luego de una cantidad fija de pasos de minimización, obtendremos un posible

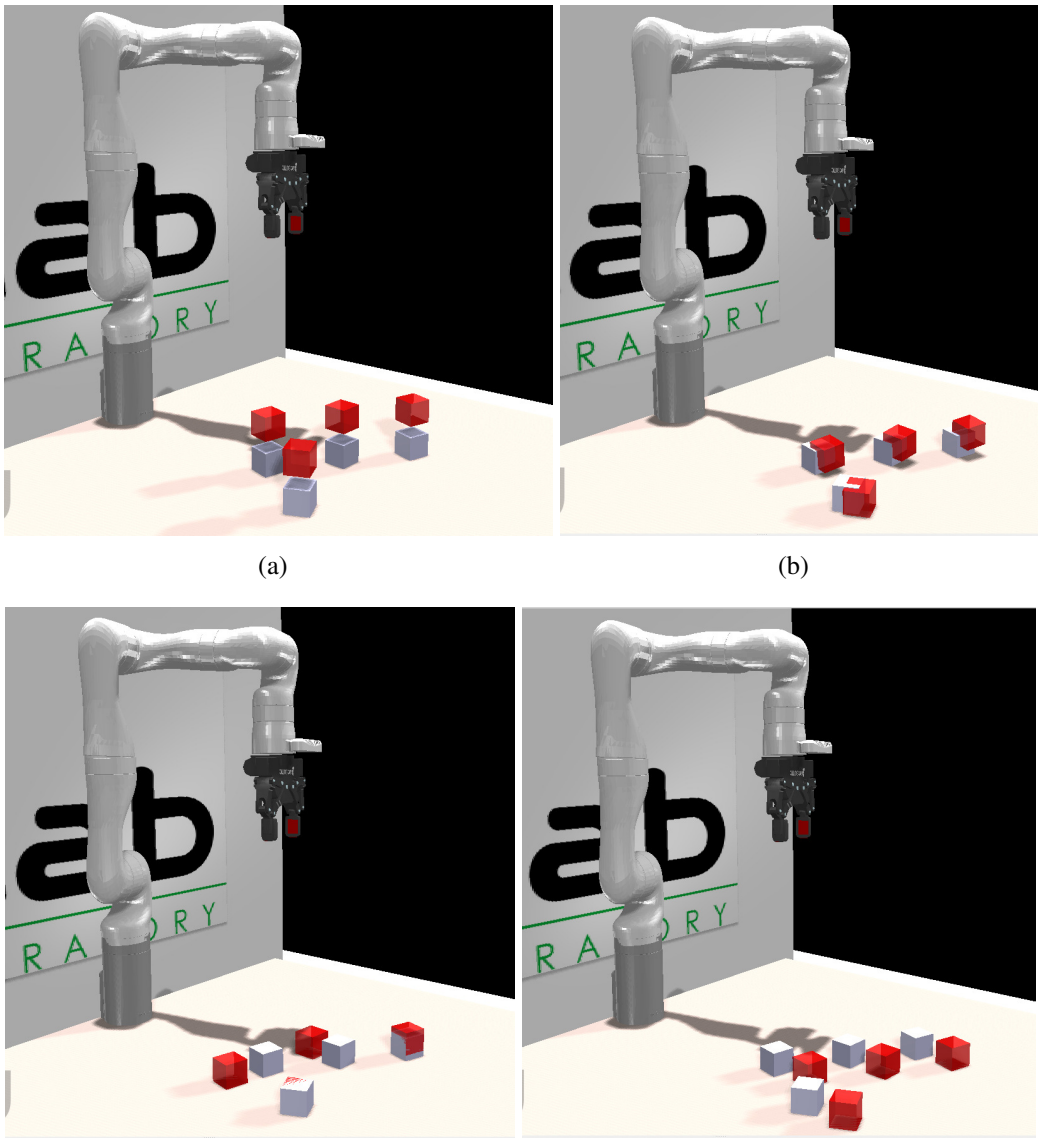
Algoritmo 1 Función de error

Input: p^r
  $DSR.publishRT(gripper, hand\_camera, p^r)$
  $C \leftarrow DSR.get\_nodes(type = Cube)$
  $aggregated \leftarrow 0$
  for $c \in C$ **do**
    $c^e \leftarrow DSR.getPosition(c)$ ▷ Siguiendo la cadena de RTs desde world
    $c^s \leftarrow DSR.getVirtualRT(c)$
    $difference \leftarrow dist(c^e, c^s)$
    $aggregated \leftarrow aggregated + difference$
  end for
  $average \leftarrow \frac{aggregated}{|C|}$
Output: $average$

valor para nuestra nueva posición de la cámara. Si esta fue capaz de eliminar el error, este nuevo valor es impactado en el sistema y tomado de aquí en adelante como la posición de la cámara.

Sin embargo, esta mejora puede ser causada por un movimiento coordinado de todos los cubos, generando la ilusión del desplazamiento del sensor, pero este caso se considera menos probable que un error en la calibración. Más aún, si este fuese el caso, el robot debería volver a detectar un error sistemático en sus percepciones y volvería a recalibrar. Por otro lado, si el error que originó el proceso de recalibración se debe a que uno o varios cubos fueron cambiados de lugar, un cambio en la posición de la cámara sería incapaz de alinear estas nuevas posiciones con las del simulador, por lo que la minimización fallaría y se mantendría la configuración previa. Estos tres casos son ilustrados en la figura 4.2, allí tanto 4.2(a) como 4.2(b) representan casos donde se encuentra un error sistemático, corregible al llevar la cámara un poco hacia la mesa o rotarla sobre su eje respectivamente. En 4.2(c) se observa un caso donde algunos cubos siguen en su lugar y otros presentan desplazamiento, este tipo de error no sistemático no puede ser corregido por una recalibración y se interpretará que los cubos simplemente cambiaron su posición. La última imagen (Figura 4.2(d)) representa un caso donde todos los cubos fueron movidos en la misma dirección y la misma distancia. En esta situación, el sistema recalibraría la posición de la cámara erróneamente, pero futuros encuentros con objetos deberían disparar una segunda etapa de calibración que corregiría esto.

Para evaluar esta funcionalidad, se utilizó una escena muy similar a la presentada en la figura 4.1. Una vez que todos los cubos son detectados e insertados en la simulación, se corrompe la cadena cinemática que conecta el mundo con la cámara.



(a) (b)

(c) (d)

Figura 4.2: Escenarios posibles para recalibración. En rojo se muestran las posiciones de los cubos reportadas por el sistema perceptivo y en blanco las previamente insertadas en el simulador. (a) y (b) representan casos de error sistemático, donde una corrección en la posición de la cámara explica la discrepancia. (c) muestra un error no sistemático, donde algunos cubos se movieron, pero no coordinadamente y (d) presenta el caso donde los cubos fueron movidos coordinadamente, indistinguible para el sistema de los casos (a) y (b).

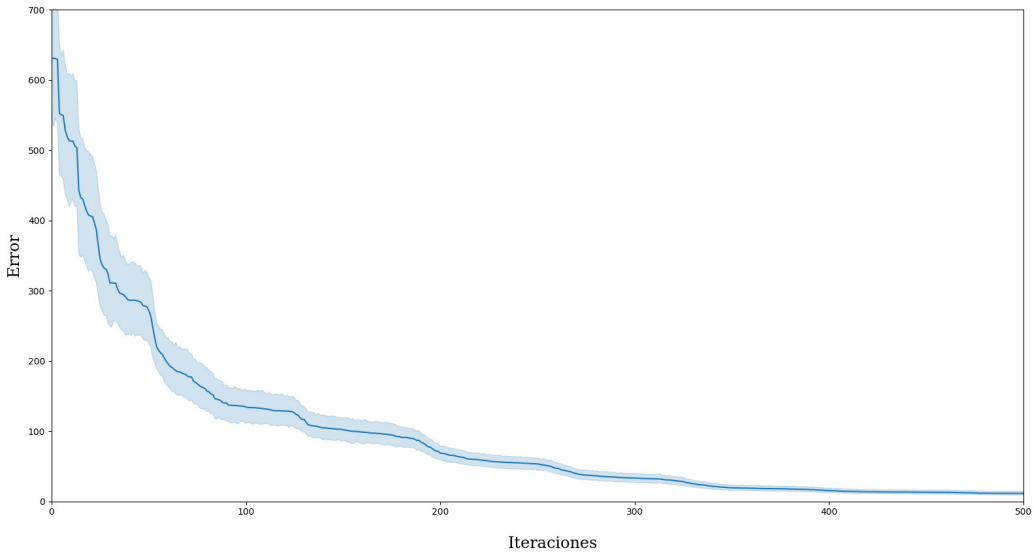


Figura 4.3: Evolución del error para las pruebas de recalibración. Se presenta la media y el intervalo de confianza de 95 %

En particular, se agrega ruido a la transformación p^r que va desde la punta del brazo al centro de la cámara. Para esto se sortean tres valores de desplazamiento de una distribución normal con media cero y desviación estándar 20 (mm) y tres valores de rotación de una distribución normal con media cero y desviación estándar 0.5 (rad) que se sumarán al valor de la transformación. Esto genera el efecto deseado, donde todos los cubos permanecen visibles pero la diferencia entre C^e y C^s es mayor de lo esperado.

Para una configuración utilizando cuatro cubos, se llevaron adelante 50 ejecuciones de este proceso. En la figura 4.3 se presenta el resultado agregado, donde podemos ver la media del error y el intervalo de confianza a 95 %. Allí se muestra cómo el error comienza en valores muy altos, pero el sistema es capaz de encontrar una transformación que lo minimice y vuelva a valores de discrepancia en un intervalo aceptable. Esta convergencia no significa por sí sola que las transformaciones encontradas sean válidas o correctas, ya que se puede haber encontrado una solución que se aleje de la realidad pero lleve el error a un mínimo (podría entenderse como un caso de *overfitting*). Sin embargo, al observar las salidas de este proceso, vemos que la convergencia también se ve en los valores y estos se encuentran cercanos a la configuración inicial (figura 4.4).

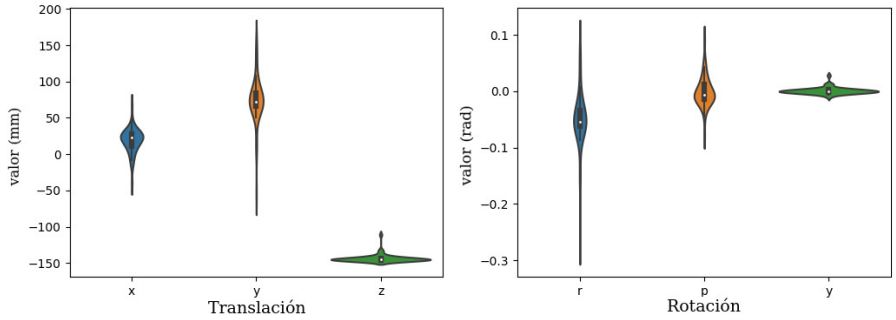


Figura 4.4: Valores de translación (x, y, z) y rotación (r, p, y) encontrados para la transformación espacial desde la punta del brazo a la cámara en las ejecuciones.

4.2.1. Posibles limitaciones de la funcionalidad

Luego de los buenos resultados obtenidos en las primeras pruebas, se decidió investigar un poco sobre las posibles limitantes para este tipo de proceso. En particular, se buscó investigar si existe una cantidad mínima de objetos para que esto funcione y si las posiciones de estos en el área de trabajo afectan la calidad de la calibración.

El procedimiento para probar las distintas variables al calibrar es similar en ambas pruebas. Primero se inicializa el sistema con un valor de p^r con una precisión baja, en particular se utiliza una precisión de un centímetro al medir cada uno de sus valores de translación. El sistema coloca los cubos que esté viendo en el simulador y se espera que este los corrija. Se calibra el sistema y se guardan los valores de la transformación propuesta.

Para evaluar cada transformación, se inicializa el sistema con su valor, se espera que coloque los cubos en la simulación y se observan los cubos desde cuatro posiciones distintas midiendo en cada una el error entre la estimación y la simulación. Utilizar varios puntos de vista evalúa la consistencia de la transformación, ya que si utilizáramos uno solo, puede que se esté ajustando a esa perspectiva. En la figura 4.5 se muestran superpuestas las posturas utilizadas en la evaluación. Para esta evaluación se utilizaron seis cubos, pero luego se descartaron algunos, debido a que presentan un nivel de error muy alto en todos los casos. Esto puede explicarse por problemas con su AprilTag o con la iluminación de ese momento (ver Anexo 1.3).

Figura 4.5: Posturas del brazo utilizadas para evaluar la consistencia en las estimaciones de las distintas RTs propuestas en las pruebas.

4.2.1.1. Cantidad de cubos

Para este experimento se calibró el sistema variando la cantidad de cubos. Para cada cantidad se realizaron tres sesiones de calibración. Se presentan los resultados combinados de estas en la figura 4.6. La línea roja punteada representa el error que comete el sistema utilizando la transformación inicial, es decir, la previa a la calibración.

Como es esperable, utilizar un solo cubo no mejora la *performance*, ya que se ajusta la transformación para que el error sea cero en la calibración, sobre ajustándose. A partir de dos cubos el sistema comienza a tener errores por debajo de los cometidos antes de ser calibrado y a partir de tres, el intervalo de confianza se mantiene casi por completo por debajo.

4.2.1.2. Posiciones de los cubos

En la prueba anterior, las posiciones de los cubos a la hora de calibrar el sistema fueron elegidos al azar utilizando un script simple. Pero se indagó también en cómo estas pueden afectar la calidad de la calibración. Para eso se diseñaron seis

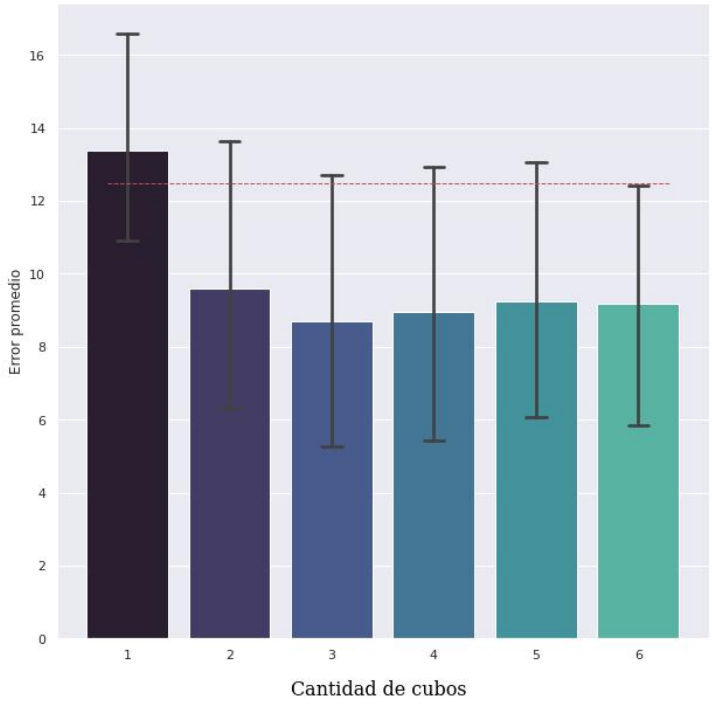


Figura 4.6: Error medio según la cantidad de cubos utilizados para la calibración. La línea punteada es el error utilizando la transformación antes de ser calibrada.

configuraciones simétricas con la intención de explorar si alguna distribución es particularmente beneficiosa.

Como se puede observar en la figura 4.7, siempre que los cubos se encuentran cubriendo toda el área de trabajo la calibración mejora las medidas con respecto a la inicial. Las configuraciones que cometen el menor error son aquellas que la cubren de forma más densa. Cuando los cubos se encuentran juntos, como en el tercer caso, la calibración resulta en un peor desempeño. Este resultado no se desvía de lo esperado. Cuando los cubos se encuentran tan cercanos la situación se asemeja a la de tener un solo cubo, el sistema puede eliminar el supuesto error sistemático al ajustarse a esta situación particular, pero generando una transformación incorrecta.

4.3. Manejo de oclusiones totales

Un problema usual para la percepción en robots son las oclusiones. Sin importar las capacidades sensoriales que el robot tenga, siempre puede sufrir de estas situaciones donde el objeto de su interés no es detectable. En nuestra situación, cuando un objeto deja de ser percibido por el agente detector, su posición directa (arista

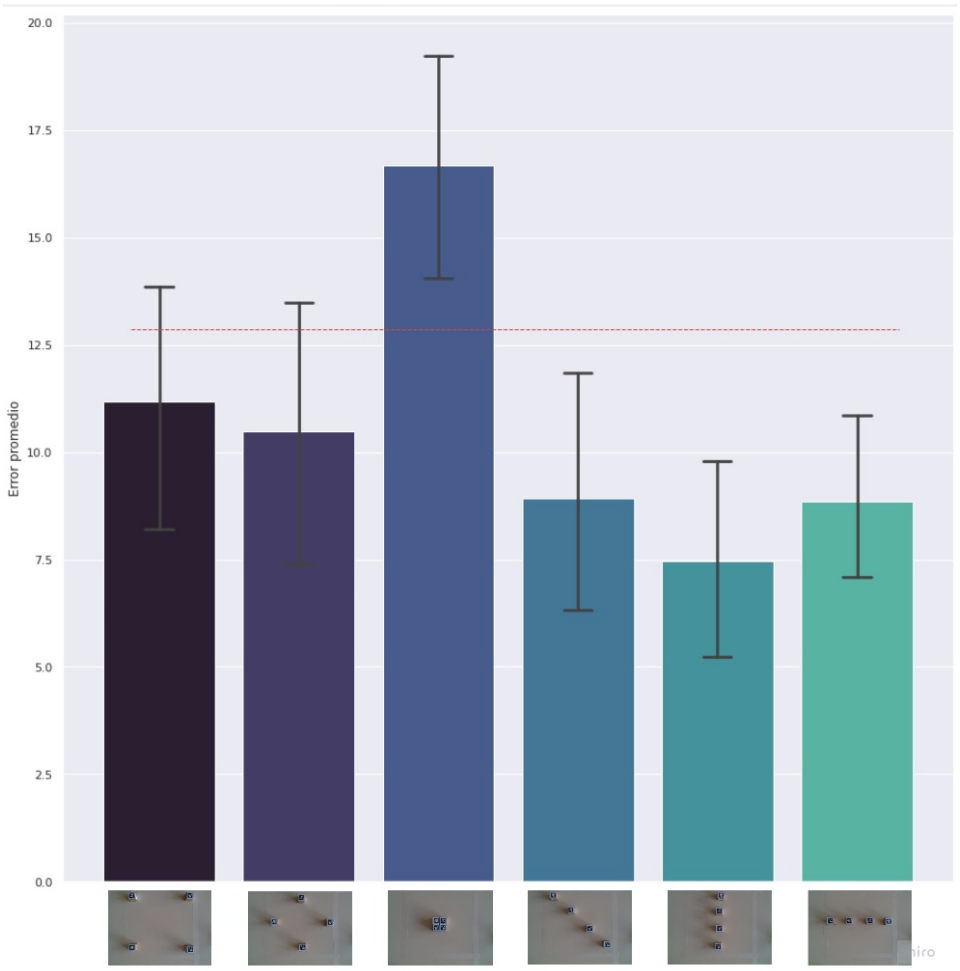


Figura 4.7: Error medio según la configuración de cubos utilizada para la calibración. La línea punteada es el error utilizando la transformación antes de ser calibrada.

RT) dejará de ser publicada y serán otros mecanismos quienes actualicen (de ser posible) su información.

Recientemente, Sallami et al. [35] identificaron este problema y mostraron un experimento muy interesante. En él, se muestra a un humano sosteniendo una pelota sobre dos cajas y colocándola dentro de una de ellas. Luego, se vuelca la caja que contiene la pelota sobre la otra, sin que en ningún momento se pueda observar directamente a la pelota. Al finalizar, se solicita al robot que indique la posición de la pelota. En su caso (al igual que en el nuestro) contar con un sistema de simulación sincronizado con el mundo hace que la respuesta sea inmediata. Al aplicar la ley de la gravedad y los algoritmos de detección de colisiones sobre la pelota en cada momento, podrá predecir correctamente que la pelota ha cambiado de contenedor sin nunca haberla percibido. Este experimento, que sirve de inspiración para los

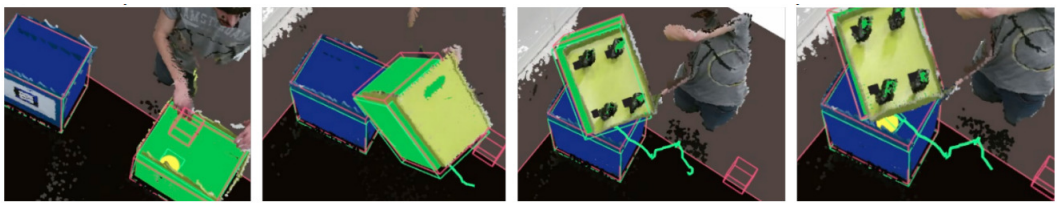


Figura 4.8: Experimento llevado adelante por Sallami et al. [35]. En él se demuestra cómo el sistema es capaz de identificar la posición de una pelota que no puede observar, utilizando un simulador. En verde se puede observar la trayectoria estimada de la pelota, que fue detectada por última vez en la posición marcada con un cubo rojo.

presentados más adelante, se muestra en la figura 4.8.

En nuestro caso, la información está siempre disponible en forma de arista *Virtual_RT* marcando dónde se estima que estará el objeto en cada momento.

4.3.1. Agente detector de manos

Cuando existe intervención humana, las leyes de la física cambian en algunas formas. En el ejemplo presentado, la caja que la persona está sosteniendo se encuentra flotando, violando la ley de la gravedad. Obviamente esto no nos sorprende, ya que nosotros entendemos que una persona es capaz de sostener en el aire un objeto y no se debe a una falla en el sistema.

Es por esto que para sus experimentos, Sallami et al. deciden identificar estas situaciones y permitir que algunos objetos floten temporalmente. Específicamente, ellos crean una máquina de estados en la cual cuando un objeto está siendo percibido en una posición que viola la gravedad (por encima de un umbral) infieren que este está siendo sostenido por una persona y permiten su posición. Esta decisión puede ser propensa a errores, por lo que en nuestro caso se desarrolló un nuevo agente para detectar estas situaciones.

Este agente es relativamente simple: toma desde la memoria de trabajo la información visual y de profundidad y utilizando MediaPipe Hands [47] detecta y localiza manos humanas en el entorno. Con esta información, genera un nodo *human_hand* y un hijo de este por cada falange de la mano (se puede configurar qué falanges son de interés). Estos nodos, conectados por aristas RT, pueden ser localizados tridimensionalmente en el mundo.

Para completar esta funcionalidad se extendió el comportamiento del agente controlador de la simulación. Cuando un nodo *finger* es insertado en la memoria de trabajo, el agente inserta en la simulación una pequeña esfera. Esta no será sometida

a la gravedad ni generará colisiones con los demás objetos, pero su posición será la misma que la insertada en el DSR en cada momento. Es con esto que el sistema es capaz de detectar cuándo una mano humana está en contacto con un objeto y es allí cuando se inserta en el sistema una arista de tipo *grasping* desde la mano al objeto que está en contacto con ella. Este tipo de razonamiento geométrico es otra de las ventajas de contar con el simulador, ya que detectar un agarre es crucial para tareas de colaboración con humanos. En respuesta a un agarre, el agente que controla la simulación marca ese objeto como no sujeto a la gravedad, permitiendo que flote. Cuando la colisión con los dedos ya no existe, la arista es borrada y el objeto vuelve a comportarse con normalidad.

4.3.2. El juego de la mosqueta

Con el objetivo de demostrar el comportamiento del sistema con estos nuevos agentes se presenta este experimento. Como se mencionó, una de las más interesantes ventajas en este caso es la capacidad de mantener localizados los objetos a pesar de no estar percibiéndolos. En su trabajo, Sallami et al. realizan una variada cantidad de experiencias demostrando esto, que fueron testeadas informalmente con nuestro sistema y se comportaron de manera similar.

En este trabajo se presentará el popular juego callejero conocido la mosqueta o *Cups and Balls*. El juego consiste de dos roles: quien mueve los objetos y quien observa. Al iniciar se presenta un objeto de interés y dos o más contenedores (suelen ser una pelota y vasos respectivamente), se coloca el objeto dentro de uno de los contenedores y comienza el juego. El objetivo de quien mueve los contenedores es hacerlo de tal manera que quien observa sea incapaz de determinar dónde estará el objeto de interés al terminar. Es esta dinámica la que hace este juego el escenario ideal para probar la capacidad de nuestro robot.

Quienes conocen el juego sabrán que normalmente este se juega de forma desleal. Quien maneja los objetos suele engañar a quienes lo observan cambiando el objeto de contenedor o directamente quitándolo de la escena. En nuestro ejemplo replicaremos este comportamiento. El experimentador tomará el rol de quien mueve los objetos y el robot deberá ser capaz de localizar el de interés en todo momento.

En la figura 4.9 se presentan algunas imágenes de lo que fue la experiencia. Primero, el robot es presentado con un cubo y dos cajas 4.9(a), cada uno de estos objetos tiene un marcador y el robot conoce la geometría de los mismos previamente, por lo que puede iniciar la escena como se ve en la figura. Cuando la persona

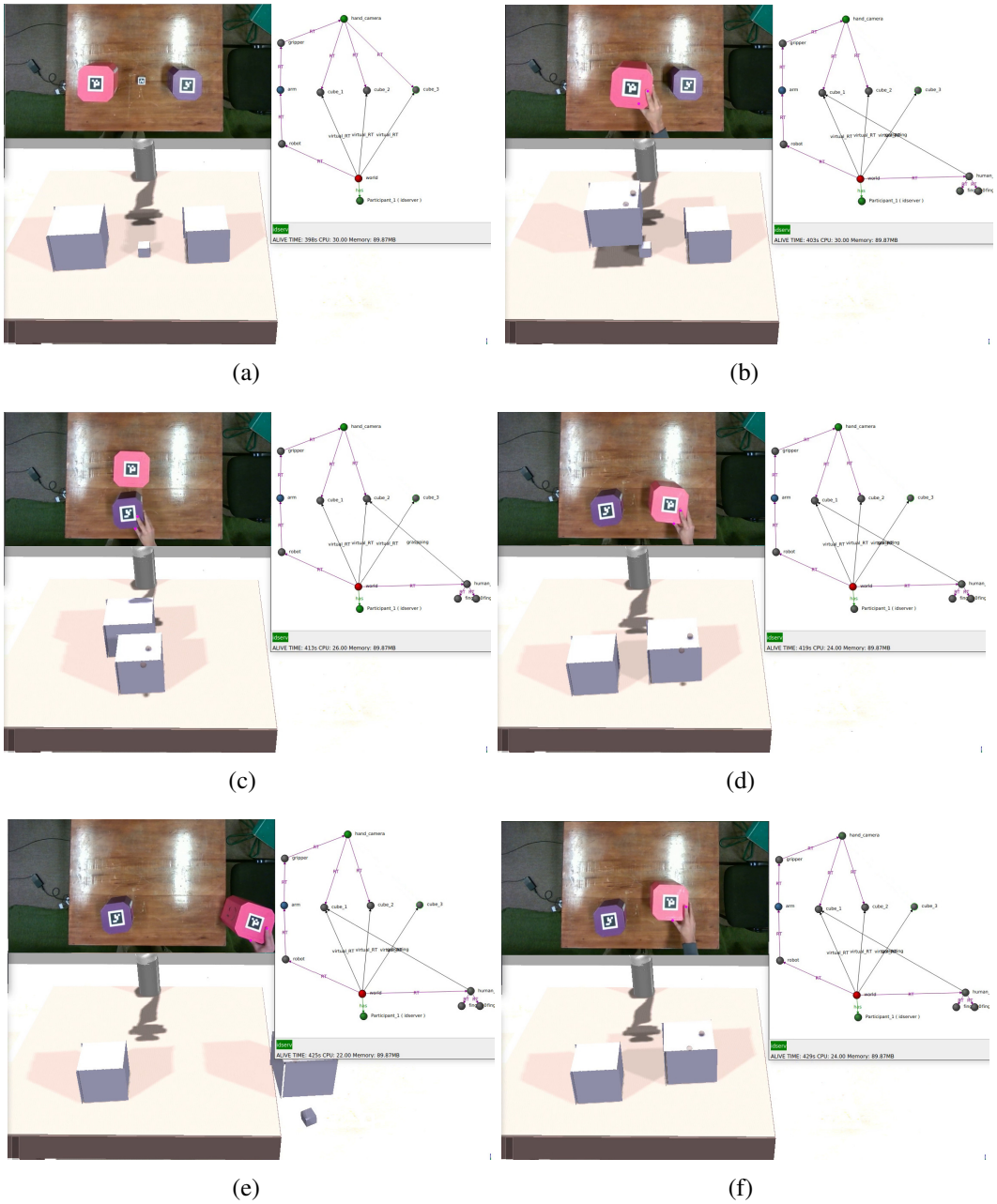
agarra una de las cajas, se identifica correctamente el agarre y aparecen en la memoria de trabajo tanto la mano como la arista *Grasping* hacia la caja (figura 4.9(b)). Se coloca la caja sobre el cubo, haciendo que este quede totalmente oculto para el robot (figura 4.9(c), nótese que ya no existe una arista RT hacia el cubo) y se intercambian las posiciones de las cajas 4.9(d). Finalmente, en la figura 4.9(e) es donde sucede el truco. El experimentador mueve la caja más allá del borde de la mesa, haciendo que el cubo caiga debajo de la misma. En esa misma imagen se puede ver que el sistema, gracias a su capacidad de simulación, puede determinar que el cubo efectivamente está cayendo.

Este experimento cualitativo apunta una vez más a mostrar el tipo de razonamiento físico que se puede obtener simplemente por sincronizar nuestra percepción con un simulador mental. Lograr identificar este tipo de situaciones utilizando solo la percepción es una tarea muy difícil, que requeriría la implementación de varias reglas o heurísticas. Cada uno de los eventos marcados en las subfiguras responde a interacciones complejas entre un agente externo y el ambiente percibido, donde aparecen varios conceptos de física intuitiva que deberían integrarse en reglas. Por ejemplo, el sistema debería identificar que si coloco un objeto en el mismo espacio físico que otro, uno debe estar "dentro" y esto es importante porque al mover el contendor, debo de alguna forma mover lo que contiene en mi representación. También debería saber que esta condición cambia si el contenedor es movido fuera de la superficie de soporte, para poder identificar que se rompe y que la caja se caería.

Es claro que no es imposible implementar estas reglas, pero lo interesante de utilizar el simulador es que no son necesarias. Tenemos en todo momento en nuestra memoria de trabajo acceso a la posición de un objeto que no es percibido desde hace varios fotogramas y que ha sufrido varias y diversas interacciones. Esto demuestra la generalidad del método presentado y es allí donde se puede ver su potencial.

4.4. La sorpresa como disparador de correcciones

Durante las pruebas con el sistema se identificaron algunas situaciones particulares en las cuales el simulador no lograba dar explicación a la percepción. Por momentos la percepción detectaba objetos en posiciones muy lejanas a las encontradas en el simulador. Y esto despierta la pregunta ¿a quién debo creer? Y la respuesta no siempre es la misma, como observador externo se pueden identificar errores en la percepción con la misma frecuencia con la que se encuentran errores de sincronización. En muchos casos, estos errores terminan convergiendo a un estado estable



(a) (b) (c) (d) (e) (f)

Figura 4.9: El juego de la mosqueta. (a) Muestra la configuración inicial. Cuando se detecta una mano, se crea su respectivo nodo y se insertan dos esferas que representan la punta de sus dedos (b). La detección de agarre puede verse por primera vez en (c) y (d) donde aparece una arista indicándolo. En (e) la caja conteniendo el cubo se mueve más allá del límite de la mesa y el simulador muestra cómo el cubo cae al suelo. (f) Presenta el estado final, donde ambas cajas vuelven a un estado indistinguible de (d), pero esta vez sin el cubo dentro.

y correcto, pero los efectos intermedios sobre objetos no percibidos pueden no ser reversibles.

Uno de los casos más comunes se da cuando un cubo se encuentra debajo de otro formando una torre y esta se mueve en conjunto a otro lugar. Cuando esto sucede, el sistema percibe el cubo de arriba, pero se encuentra constantemente en un bucle donde el simulador hace que el cubo caiga cuatro centímetros, pero la percepción vuelve a detectarlo en el aire. Esta discrepancia podría ser detectada como sorprendente y solucionada utilizando información semántica sobre el estado del mundo, como puede ser una arista *On*, que indique que anteriormente el cubo que está flotando se encontraba sobre otro.

4.4.1. Agente de sorpresa

Estos casos donde mi percepción no coincide con mi modelo mental, han sido ampliamente estudiados en cognición. En el trabajo de Smith et al. [38] se le llama a esto violación de expectativa, siendo que una situación particular observada no coincide con nuestra expectativa según la física intuitiva. Esta señal de sorpresa puede ser una forma de detectar errores, pero por las mismas razones, es una oportunidad para intentar corregir nuestro modelo.

Inspirado en esta idea, este agente se encarga de monitorear el estado de la memoria de trabajo y en caso de ser necesario, disparar una señal de sorpresa. En particular, este agente calcula en cada momento la diferencia entre la posición de un objeto según la percepción (siguiendo la cadena cinemática hasta este) y el modelo mental (representado por las aristas *Virtual_RT*). Si esta diferencia supera un umbral, el agente modifica el atributo *Surprising* dentro de este cubo, indicando que se encuentra en violación de las expectativas del sistema.

4.4.2. Modificaciones al agente manejador de la simulación

Teniendo esta nueva información sobre cubos que puede que se encuentren en violación de la expectativa del sistema, el agente que maneja la simulación será el encargado de solucionar estas discrepancias siempre que sea posible. Para los efectos de esta prueba, nos centraremos en el caso descrito anteriormente, donde se cuenta con información sobre algunos cubos que se encuentran sobre otros.

El agente ahora reacciona a cambios en el atributo *Surprising* de cualquiera de los cubos. Para intentar devolver el sistema a un estado estable se utiliza la información semántica para probar algunas explicaciones. Si un cubo es sorprendente

y según la memoria de trabajo, debería tener otro debajo, se intenta explicar esta situación colocándolos una vez más de esa forma. En el caso contrario, si estoy percibiendo un cubo que se supone está debajo de otro, significa que esto ya no se cumple dado que sería imposible haberlo percibido (Algoritmo 2).

Algoritmo 2 Procesamiento de cubos sorprendentes

Input: $S \leftarrow \{c_i | c_i \in C, c_i.surprising = True\}$
  for $c \in S$ **do**
    if $\exists c_2 : on(c_2, c)$ **then**
      $c_2.position \leftarrow$ under c
      simlation.update_position(c_2)
    else if $\exists c_2 : on(c, c_2)$ **then**
      dsr.delete ($on(c, c_2)$)
      simlation.update_position(c)
    else
      simlation.update_position(c)
    end if
  end for

Este simple proceso que toma en cuenta la información contenida en la memoria de trabajo ha demostrado ser suficiente para mantener el estado de la simulación sincronizado en los casos estudiados a continuación.

4.4.3. Agente planificador

Una vez desarrollada la funcionalidad de respuesta a eventos sorprendentes, se decidió implementar una clásica tarea de manipulación de cubos para ponerla a prueba. La idea es que el robot sea capaz de identificar el estado en el que se encuentran los cubos, se le dará un estado final deseado y deberá ser capaz de planificar y ejecutar los pasos necesarios para conseguirlo. Para esto se desarrolló un nuevo agente, quien llevará adelante esta tarea.

Antes de planificar los pasos necesarios para alcanzar un estado objetivo, debemos dar un paso atrás y definir cómo representaremos el mundo en el que el robot se desarrollará y las dinámicas que lo gobiernan. En este caso, se decidió utilizar el estándar para este tipo de tareas: el lenguaje PDDL (del inglés *Planning Domain Definition Language*) [12]. Allí se deben describir los predicados, las acciones y las consecuencias de estas. Las principales acciones que puede tomar el brazo para este ejemplo son (i) tomar un cubo y (ii) colocarlo sobre otro. Para ver la definición completa pueden referirse al anexo 1.2.

Al iniciarse, este agente tomará la información disponible en la memoria de trabajo para computar el estado inicial. En particular, tendrá en cuenta las aristas existentes entre los cubos, como pueden ser las aristas *On*, que indican que un cubo está apoyado sobre otro. Esta es otra ventaja de utilizar la arquitectura CORTEX, donde podemos incluir aristas semánticas para representar cambios que se hagan en el entorno.

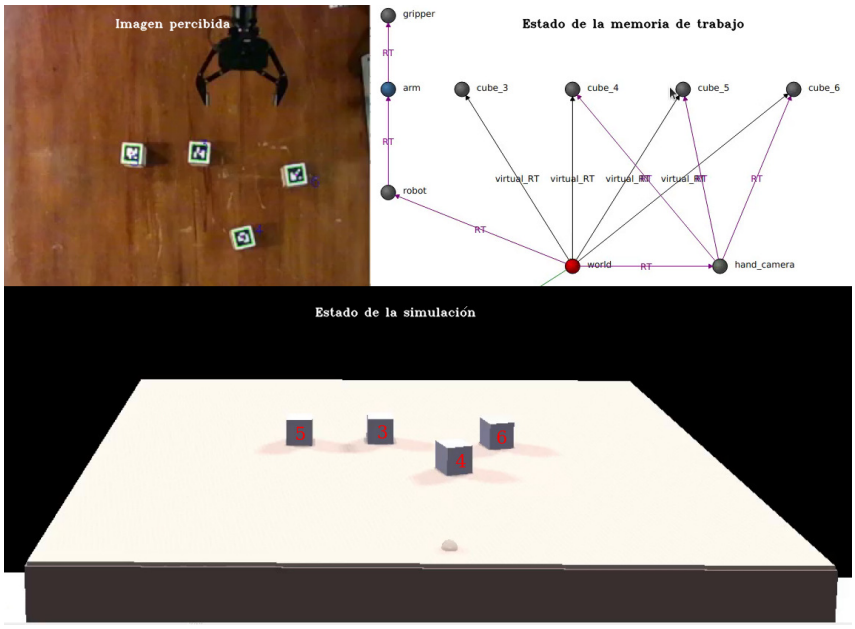
Además de planificar, es este agente el encargado de ejecutar el plan que construyó. Una vez que se obtiene la secuencia de acciones que convertirá el estado actual en el final, este agente enviará las posiciones al brazo para que este las ejecute una a una e impactará en el DSR sus efectos (por ejemplo insertar una arista *On* luego de colocar un cubo sobre otro). Sin embargo, en cualquier momento de la ejecución puede surgir algún imprevisto. Si por alguna razón el estado del mundo cambia en la memoria de trabajo sin ser esto obra propia, el agente tomará este estado como inicial y replanificará, para seguir son su misión en caso de ser posible.

4.4.4. Desarrollo de la prueba

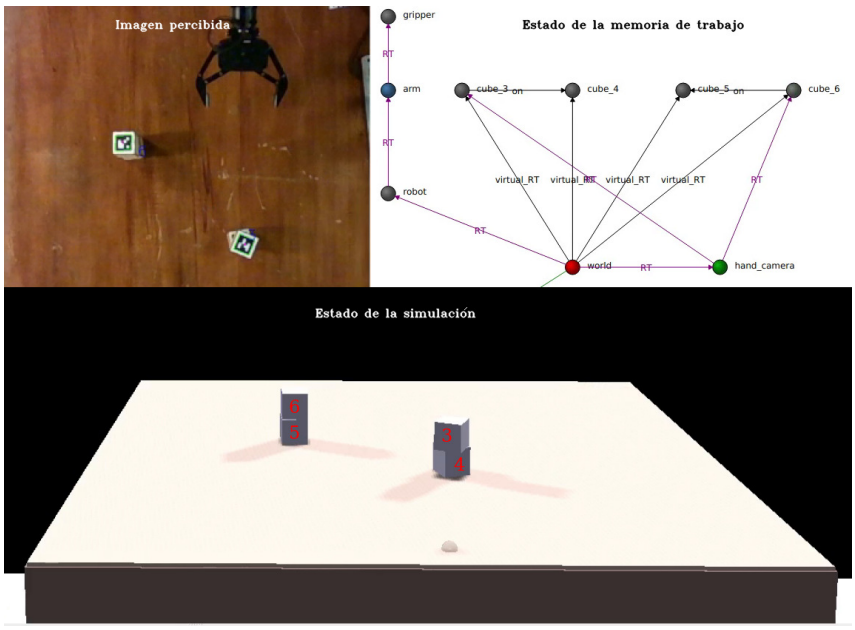
En la prueba donde se integraron estos nuevos agentes, se inicia con los cubos tres, cuatro, cinco y seis colocados sobre la mesa (figura 4.10(a)) y se le indica al agente planificador que el estado final presenta el cubo tres sobre el cuatro y el seis sobre el cinco (figura 4.10(b)). Es importante notar que esta vez la cámara se encuentra en un marco, independiente del robot. Es por esto que se puede ver que su posición está determinada por una arista RT directamente desde el nodo *world*. Esta situación genera que el sistema pueda percibir cubos que están agarrados por el brazo como flotando en el aire, pero el agente planificador insertará aristas *grasping* cuando ejecute la acción de tomar un cubo y, tal como en los ejemplos anteriores, el sistema dejará de aplicar la gravedad a este.

En esta prueba no se encuentran en ejecución ni el agente de calibración ni el de detección de manos.

Resolver la tarea original de apilar dos torres de cubos no es necesariamente un desafío complicado, lo que se busca del sistema en esta instancia es que sea capaz de lograrlo incluso sufriendo perturbaciones mientras lo hace. La evaluación de esta funcionalidad es completamente cualitativa, donde la tarea del experimentador es modificar el entorno para confundir al robot mientras este intenta adaptarse en tiempo real. A continuación se presentan cuatro pruebas unitarias donde se muestra



(a)



(b)

Figura 4.10: Estados inicial (a) y final (b) para la prueba de planificación.

el desempeño del robot.[1]

4.4.4.1. Movimiento de torre completa

La primer prueba responde directamente al ejemplo implementado. Esta consiste en mover una torre de dos cubos y evaluar si el sistema es capaz de reconocer esta perturbación y utilizar la información semántica del grafo para corregirla.

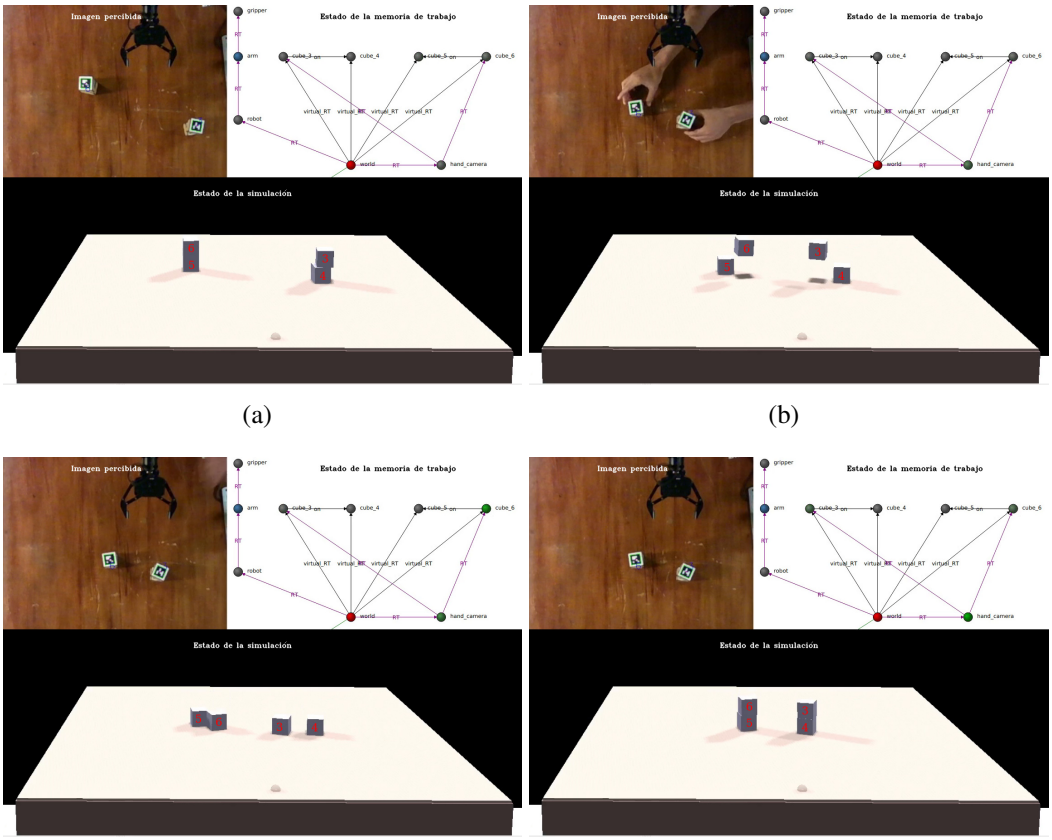
En la figura 4.11 se presentan capturas de la ejecución de esta perturbación. Como se puede observar, el sistema mantiene el estado final bien representando en la figura 4.11(a) hasta que el experimentador mueve ambas torres de lugar (figura 4.11(b)). Estas nuevas posiciones disparan una señal de sorpresa, ya que no hay coincidencia entre percepción y simulación 4.11(c). Consultando en la memoria de trabajo, el sistema intenta explicar esta situación colocando los cubos cuatro y cinco por debajo de los cubos tres y seis respectivamente 4.11(d). Esta modificación resulta en un estado estable, por lo que se toma como la explicación de la situación.

4.4.4.2. Corrección de errores en la simulación

No es extraño que durante la experimentación alguna falla en el posicionamiento de los cubos genere situaciones erróneas. Un caso común se da cuando un cubo agarrado choca contra otro que no está siendo percibido en ese momento. El problema surge porque el cubo agarrado no responde a las físicas y al chocar con otro puede aplicarle una gran fuerza (ya que es el segundo quien absorbe toda la reacción). Cuando este segundo cubo no está siendo percibido, el sistema no puede volverlo a su lugar y el modelo queda en un estado inconsistente.

Un ejemplo de esta situación es presentado en la figura 4.12. Mientras el brazo estaba colocando un cubo sobre el otro (figura 4.12(a)) un error en el posicionamiento genera que el de abajo salga disparado (figura 4.12(b)). Cuando esto sucede y el cubo ya no está siendo agarrado, se puede ver en la simulación que este cae sobre la mesa, pero esta posición no coincide con la percepción y dispara una señal de sorpresa (figura 4.12(c)). Una vez más, gracias a la información semántica, se coloca el cubo debajo del otro y se consigue una configuración estable 4.12(d). Esto muestra que con muy poca información sobre el estado del mundo se pueden corregir varias situaciones distintas.

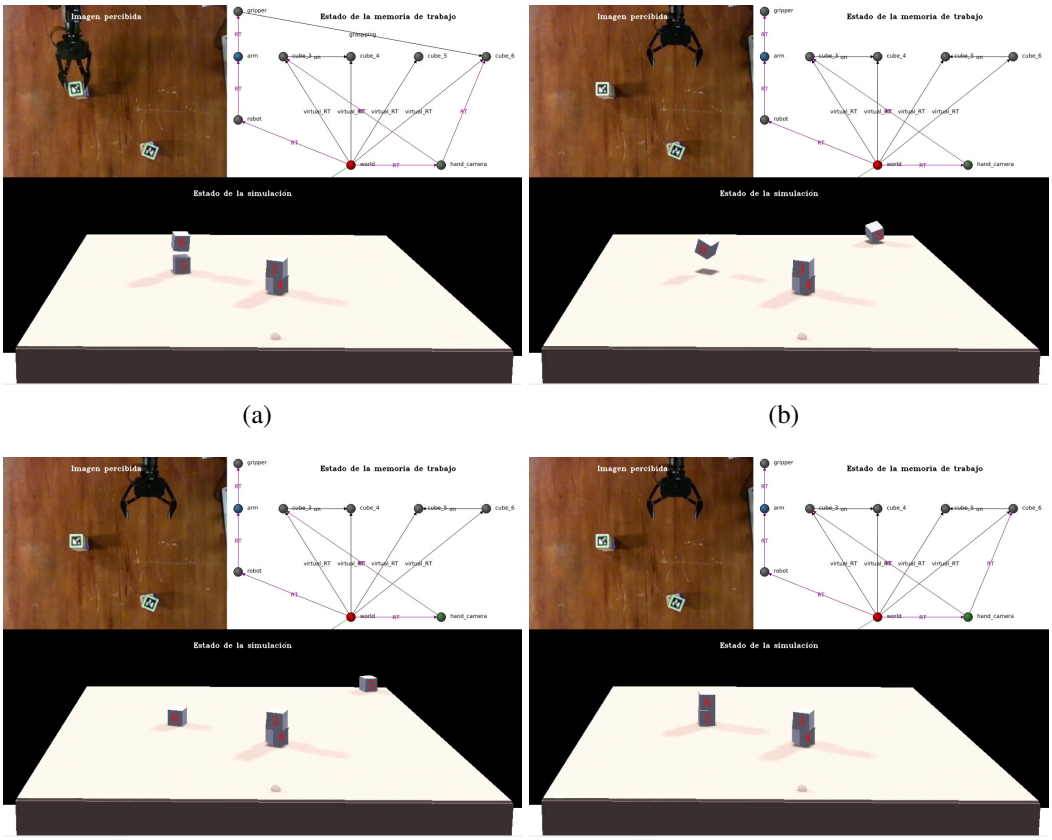
[1] Los videos asociados a cada prueba se encuentran en la sección *Maestría* de la página https://www.fing.edu.uy/~gtrinidad



(a) (b)

(c) (d)

Figura 4.11: Ejecución de la prueba de movimiento de torre completa.



(a) (b)

(c) (d)

Figura 4.12: Ejemplo de corrección basada en información semántica.

4.4.4.3. Desarmado de torre

En este ejemplo veremos el otro flujo de comportamiento del sistema. En este caso el experimentador desarma una torre en la mitad del plan, haciendo que el robot reaccione, resincronice y vuelva a planificar.

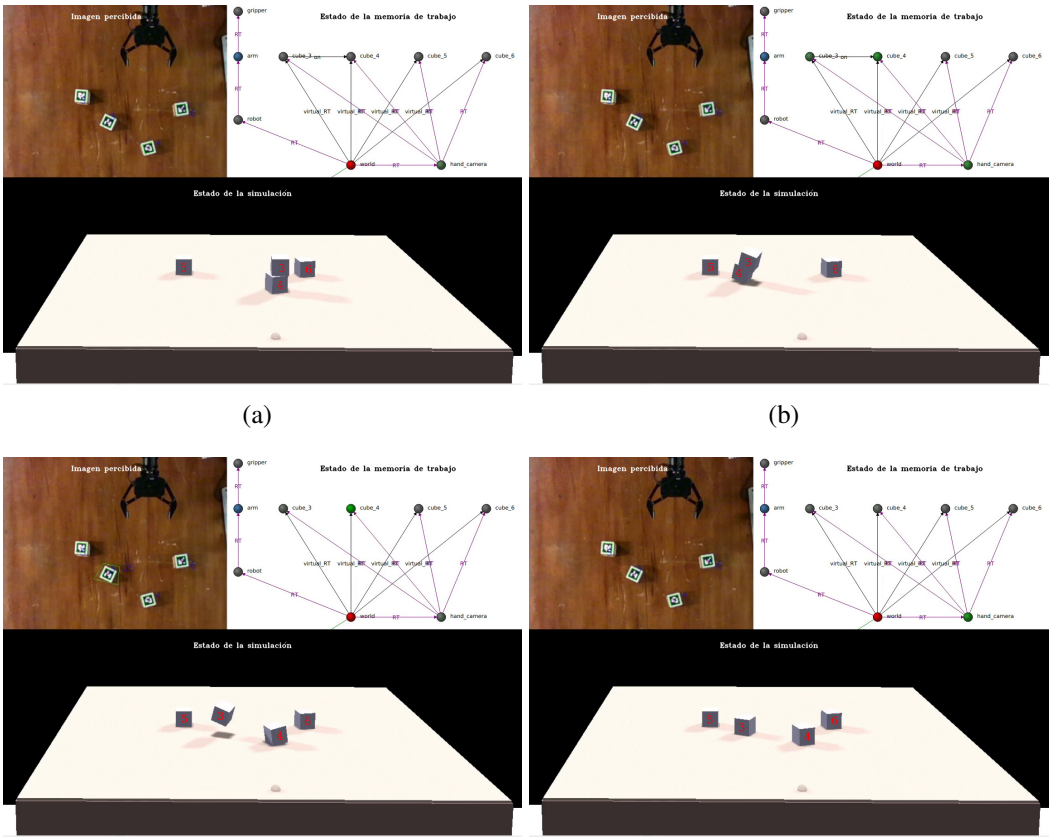
La figura 4.13 presenta la ejecución de la prueba. Como parte de su misión, el robot ya había formado la primer torre, colocando el cubo tres sobre el cuatro. Pero antes que pueda ir por la segunda, el experimentador vuelve a colocar el cubo tres en la mesa (figura 4.13(a)). Percibir el cubo tres en este nuevo lugar dispara la señal de sorpresa y como este cubo se encontraba sobre el cuatro, el sistema lo coloca por debajo para probar si eso soluciona la discrepancia (figura 4.13(b)). Como se observa en la figura 4.13(c), es ahora el cubo cuatro que genera la sorpresa, ya que al ponerlo debajo del tres su posición discrepa con la percibida. El sistema, como reacción a esto, borra la arista que conecta los dos cubos, logrando volver a un estado armonioso entre la percepción y el modelo (figura 4.13(d)).

No se presenta en esta figura, pero este cambio dispara una replanificación indicando que el robot debe volver a montar esta primer torre.

4.4.4.4. Torre de tres cubos

Como prueba final y a modo de explorar la reacción del sistema a situaciones fuera de su planificación se diseña este simple experimento. Ahora el robot ya ha terminado de montar las dos torres (figura 4.14(a)) y el experimentador coloca el cubo seis sobre el tres (figura 4.14(b)). Al detectar este cubo fuera de la posición esperada y conociendo su relación con el cubo cinco, se intenta explicar la nueva posición colocando el cinco debajo (figura 4.14(c)). De forma similar a lo que sucede en la prueba anterior, la presencia del cubo cinco allí discrepa con la percepción y, por lo tanto, se lo posiciona donde es percibido y se borra la arista que lo conecta con el seis (figura 4.14(d)).

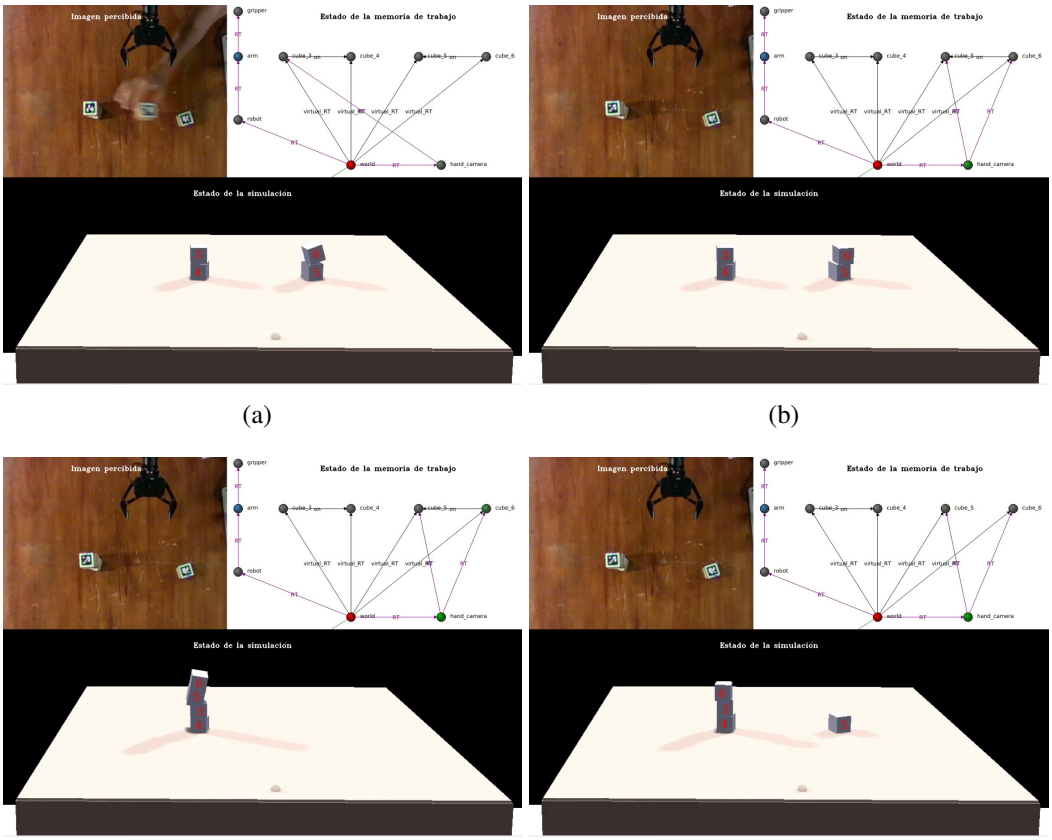
El estado en que queda la simulación se corresponde perfectamente con la realidad, pero la memoria de trabajo no captura completamente las relaciones entre los cubos. Con este estado, el robot puede tomar cualquiera de los cubos ya que las posiciones que contiene para ellos son correctas, pero no podrá corregir la simulación ante algunas perturbaciones particulares (como mover toda la torre de tres en su conjunto).



(a) (b)

(c) (d)

Figura 4.13: Ejecución de la prueba de desarmado de torre.



(a) (b)

(c) (d)

Figura 4.14: Ejecución de la prueba de torre de tres cubos.

Capítulo 5

Conclusiones y trabajos a futuro

La robótica ha tomado históricamente como parte de su inspiración los hallazgos de la biología, reproduciendo comportamientos, formas y estrategias en búsqueda de mejores robots. Hoy en día la computación avanza a un ritmo acelerado y se ha generado una doble interacción entre esta área y los últimos hallazgos en ciencias cognitivas. En esta modalidad, la implementación de modelos computacionales permite probar teorías cognitivas y a la vez, de ser efectivas, mejoran las capacidades de nuestros agentes. En este trabajo se tomó como inspiración la teoría de la física intuitiva como un simulador en nuestra mente, creando una implementación de esta y probando los beneficios que puede traernos.

Se realizaron una serie de pruebas para demostrar algunas de las capacidades que emergen al utilizar este modelo. En las primeras, al igual que en trabajos previos, se pudo observar que las reglas físicas del simulador son útiles para corregir la posición de algunos objetos detectados. En particular, errores en la estimación de la distancia son corregidos al aplicar la gravedad y al evitar que los objetos se superpongan. Más allá del aumento de la coherencia y la precisión, resulta interesante remarcar lo automático del enfoque: estas correcciones son obtenidas sin la necesidad de agregar reglas y son independientes del objeto que se quiera utilizar, característica que se mantendrá en el resto de los resultados. A partir de estas correcciones surge un evento interesante, donde la utilización de una memoria de trabajo permite detectar situaciones donde la percepción parece no coincidir con la física intuitiva. Esto dio lugar a las siguientes pruebas, donde gracias a este mecanismo de corrección se implementó un módulo capaz de tomar estas discrepancias y re calibrar las cadenas cinemáticas propias del robot. Los resultados no pretenden ser estadísticamente exhaustivos, pero resultan positivos y despiertan un interesan-

te camino a seguir, donde las intuiciones puedan detectar errores de calibración y corregirlos durante la ejecución de las tareas del robot.

Otra de las posibilidades que brinda la sincronización de la percepción con un mundo simulado es reportada en la prueba sobre oclusiones totales. En el juego de la mosqueta, el robot es capaz de mantener una estimación relativamente precisa sobre la posición de un cubo que no percibe durante varios minutos. Una vez más, esto se obtiene sin el despliegue de nuevos agentes y sin la necesidad de reglas o aprendizajes. Reportando solamente los cambios que si son percibidos, podemos utilizar este motor de físicas intuitivas para, calculando las interacciones con objetos previamente observados, mantener una representación del mundo físicamente plausible. La representación obtenida, sin embargo, puede no corresponderse con la percepción, dando lugar a situaciones sorprendentes para el sistema. Para estudiar estos eventos se llevaron adelante nuevos experimentos, donde el robot debía resolver estas diferencias utilizando su memoria de trabajo e información semántica sobre los cubos. Mientras intentaba construir dos torres, se perturbaba la posición de los cubos de diversas formas: moviendo los cubos, moviendo las torres e incluso desarmándolas. Utilizando un simple predicado que indicaba si un cubo había sido colocado sobre otro, se consiguió en todos los casos conciliar la percepción con la memoria de trabajo y el simulador. Para esto se diseñaron un conjunto de acciones donde el sistema impactaba la hipótesis más simple en el simulador, intentando dar sentido a lo percibido. La eficacia con la que se comportó el sistema utilizando estas simples reglas es prometedora y presenta un nuevo camino donde se utilizan las físicas para alimentar la descripción semántica de una escena.

El objetivo de este trabajo no es proponer los mecanismos exactos utilizados como implementación de un sentido común en robots. Se busca resaltar cómo embeber un simulador como parte nativa del *pipeline* de procesamiento del robot abre varias posibilidades de mejora sin la necesidad de desarrollar software específico ni entrenar grandes modelos de aprendizaje. Estas características son cruciales, ya que tanto el desarrollo como el entrenamiento de algoritmos es un proceso costoso y normalmente poco generalista. Si queremos que los robots puedan desempeñarse en entornos desestructurados y cambiantes, es necesario que cuenten con técnicas independientes de estos, reutilizables y con un propósito lo más general posible.

Esto no implica que las técnicas de aprendizaje no puedan utilizarse. Por ejemplo, en nuestro trabajo se agregó un proceso para desambiguar eventos sorprendentes donde las posibilidades fueron creadas manualmente. Un agente podría, a través de su experiencia propia, generar una batería de explicaciones que consultar. Por

otro lado, durante la experimentación se identificaron patrones repetitivos sobre la señal de sorpresa que se asocian directamente a algunas situaciones particulares que no fueron investigadas y podrían ser explotadas por un agente clasificador. También existen trabajos basados en redes neuronales que logran estimar propiedades físicas como masas y centros de gravedad que podrían mejorar las estimaciones realizadas por el simulador [46]. Al mismo tiempo, la capacidad de describir el mundo en objetos y posiciones físicamente correctas puede acelerar los tiempos necesarios para el aprendizaje. Si se busca que un agente encuentre políticas directamente de su entrada sensorial, este deberá primero aprender a interpretarla para luego darse a la tarea de cumplir su misión. Utilizando el nivel de abstracción que la sincronización con el simulador nos da, se puede evitar este proceso y permitir un aprendizaje a más alto nivel que requeriría menos episodios de interacción con el entorno.

A partir de los resultados obtenidos en esta investigación, se crean dos posibles caminos a seguir. El primero es extender estas implementaciones para explotar al máximo las ventajas que nos dan. Esta línea se beneficiaría de agentes que detecten objetos realistas, que incluyan más información semántica o que incluyan pruebas físicas más relevantes como estabilidad de agarres o de apoyos. Incluir incertidumbre en la simulación es definitivamente un camino a seguir para poder realizar proyecciones temporales y planificar acciones utilizando las físicas [4][27].

El otro camino involucra profundizar más en los conceptos cognitivos de la física intuitiva para incluirlos en este *pipeline*. Resulta relevante investigar cómo un agente puede aprender un simulador, construyendo modelos sobre los comportamientos de objetos y agentes a su alrededor, ya que la génesis de este sigue sin ser clara. Aquí se necesitará combinar al máximo las investigaciones en ciencias cognitivas con los modelos desarrollados para descubrir y replicar esta increíble capacidad.

El área del *Machine Common Sense* se encuentra en su infancia [13]. Hoy en día nos encontramos probando las primeras ideas sobre robots reales, donde observaremos similitudes y diferencias e intentaremos llegar a un consenso. Este trabajo presenta una implementación concreta de la teoría del motor de videojuegos en tu mente sobre una arquitectura cognitiva y demuestra la gran variedad de ventajas que este mecanismo puede traer a los robots autónomos. Este tipo de desarrollos son un argumento sobre la importancia y potencialidad de estas nuevas ideas y busca inspirar nuevas investigaciones que nos lleven hacia robots cada vez más inteligentes.

Referencias bibliográficas

[1] Alan Baddeley. "Working memory". En: *Science* 255.5044 (1992), pp. 556-559.

[2] Alan Baddeley. "Working memory: Theories, models, and controversies". En: *Annual review of psychology* 63 (2012), pp. 1-29.

[3] Andrew G BartoSridhar Mahadevan. "Recent advances in hierarchical reinforcement learning". En: *Discrete event dynamic systems* 13.1-2 (2003), pp. 41-77.

[4] Peter W Battaglia, Jessica B HamrickJoshua B Tenenbaum. "Simulation as an engine of physical scene understanding". En: *Proceedings of the National Academy of Sciences* 110.45 (2013), pp. 18327-18332.

[5] Bernd Winkler. *April Tag Detector*. 7 de diciembre de 2018. URL: https://pypi.org/project/apriltag/.

[6] Rodney A Brooks. "Artificial life and real robots". En: *Proceedings of the First European Conference on artificial life*. 1992, pp. 3-10.

[7] Pablo Bustos, Luis Jesús Manso, Antonio J Bandera, Juan P Bandera, Ismael Garcia-VareaJesus Martinez-Gomez. "The CORTEX cognitive robotics architecture: Use cases". En: *Cognitive Systems Research* 55 (2019), pp. 107-123.

[8] David Coleman, Ioan Sucan, Sachin ChittaNikolaus Correll. "Reducing the barrier to entry of complex robotic software: a moveit! case study". En: *arXiv preprint arXiv:1404.3785* (2014).

[9] CoppeliaRobotics. *ZMQ REMOTE API*. URL: https://github.com/CoppeliaRobotics/zmqRemoteApi.

[10] Erwin CoumansYunfei Bai. "Pybullet, a python module for physics simulation for games, robotics and machine learning". En: (2016).

[11] Eprosima. *Fast DDS*. 9 de enero de 2023. URL: https://www.eprosima.com/index.php/resources-all/performance/eprosima-fast-dds-performance.

[12] M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. WeldD. Wilkins. *PDDL—The Planning Domain Definition Language*. 1998. URL: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.37.212.

[13] David Gunning. "Machine common sense concept paper". En: *arXiv preprint arXiv:1810.07528* (2018).

[14] Simon Hangl, Vedran Dunjko, Hans J BriegelJustus Piater. "Skill learning by autonomous robotic playing using active learning and creativity". En: *arXiv preprint arXiv:1706.08560* (2017).

[15] Kinova Robotics. *Kortex API*. 20 de noviembre de 2022. URL: https://github.com/Kinovarobotics/kortex.

[16] Katherine D KinzlerElizabeth S Spelke. "Core systems in human cognition". En: *Progress in brain research* 164 (2007), pp. 257-264.

[17] George KonidarisAndrew Barto. "Skill discovery in continuous reinforcement learning domains using skill chaining". En: *Advances in neural information processing systems* 22 (2009).

[18] Iuliia Kotseruba, Oscar J Avella GonzalezJohn K Tsotsos. "A review of 40 years of cognitive architecture research: Focus on perception, attention, learning and applications". En: *arXiv preprint arXiv:1610.08602* (2016), pp. 1-74.

[19] James R Kubricht, Keith J HolyoakHongjing Lu. "Intuitive physics: Current research and controversies". En: *Trends in cognitive sciences* 21.10 (2017), pp. 749-759.

[20] Adam Lerer, Sam GrossRob Fergus. "Learning physical intuition of block towers by example". En: *International conference on machine learning*. PMLR. 2016, pp. 430-438.

[21] Patrick Mania, Franklin Kenghagho Kenfack, Michael NeumannMichael Beetz. "Imagination-enabled robot perception". En: *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2021, pp. 936-943.

[22] Luis Manso, Pilar Bachiller, Pablo Bustos, Pedro Núnez, Ramón CintasLuis Calderita. “Robocomp: a tool-based robotics framework”. En: *International Conference on Simulation, Modeling, and Programming for Autonomous Robots*. Springer. 2010, pp. 251-262.

[23] George A Mashour, Pieter Roelfsema, Jean-Pierre ChangeuxStanislas Dehaene. “Conscious processing and the global neuronal workspace hypothesis”. En: *Neuron* 105.5 (2020), pp. 776-798.

[24] Matthew T Mason. “Toward robotic manipulation”. En: *Annual Review of Control, Robotics, and Autonomous Systems* 1 (2018), pp. 1-28.

[25] Warren S McCullochWalter Pitts. “A logical calculus of the ideas immanent in nervous activity”. En: *The bulletin of mathematical biophysics* 5 (1943), pp. 115-133.

[26] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan WierstraMartin Riedmiller. “Playing atari with deep reinforcement learning”. En: *arXiv preprint arXiv:1312.5602* (2013).

[27] Lorenz MösenlechnerMichael Beetz. “Fast temporal projection using accurate physics-based geometric reasoning”. En: *2013 IEEE International Conference on Robotics and Automation*. IEEE. 2013, pp. 1821-1827.

[28] Roozbeh Mottaghi, Mohammad Rastegari, Abhinav GuptaAli Farhadi. ““What happens if...” Learning to Predict the Effect of Forces in Images”. En: *European conference on computer vision*. Springer. 2016, pp. 269-285.

[29] Edwin Olson. “AprilTag: A robust and flexible visual fiducial system”. En: *2011 IEEE international conference on robotics and automation*. IEEE. 2011, pp. 3400-3407.

[30] Sergiu Oprea, Pablo Martinez-Gonzalez, Alberto Garcia-Garcia, John Alejandro Castro-Vargas, Sergio Orts-Escolano, Jose Garcia-RodriguezAntonis Argyros. “A review on deep learning techniques for video prediction”. En: *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2020).

[31] Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H MillerSebastian Riedel. “Language models as knowledge bases?” En: *arXiv preprint arXiv:1909.01066* (2019).

[32] David PremackGuy Woodruff. “Does the chimpanzee have a theory of mind?” En: *Behavioral and brain sciences* 1.4 (1978), pp. 515-526.

[33] Intel RealSense. *PyRealSense 2*. URL: https://github.com/IntelRealSense/librealsense.

[34] Eric Rohmer, Surya PN SinghMarc Freese. "V-REP: A versatile and scalable robot simulation framework". En: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE. 2013, pp. 1321-1326.

[35] Yoan Sallami, Séverin Lemaignan, Aurélie ClodicRachid Alami. "Simulation-based physics reasoning for consistent scene estimation in an HRI context". En: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2019, pp. 7834-7841.

[36] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot et al. "Mastering the game of Go with deep neural networks and tree search". En: *nature* 529.7587 (2016), pp. 484-489.

[37] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel et al. "Mastering chess and shogi by self-play with a general reinforcement learning algorithm". En: *arXiv preprint arXiv:1712.01815* (2017).

[38] Kevin Smith, Lingjie Mei, Shunyu Yao, Jiajun Wu, Elizabeth Spelke, Josh TenenbaumTomer Ullman. "Modeling expectation violation in intuitive physics with coarse probabilistic object representations". En: *Advances in neural information processing systems* 32 (2019).

[39] Kevin A SmithEdward Vul. "Sources of uncertainty in intuitive physics". En: *Topics in cognitive science* 5.1 (2013), pp. 185-199.

[40] Elizabeth S Spelke, Karen Breinlinger, Janet MacomberKristen Jacobson. "Origins of knowledge." En: *Psychological review* 99.4 (1992), p. 605.

[41] Richard S SuttonAndrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[42] Guillermo Trinidad Barnech, Gonzalo Tejera, Juan Valle-Lisboa, Pedro Núñez, Pilar BachillerPablo Bustos. "Initial Results with a Simulation Capable Robotics Cognitive Architecture". En: *Iberian Robotics conference*. Springer. 2023, pp. 612-623.

[43] Tomer D Ullman, Elizabeth Spelke, Peter BattagliaJoshua B Tenenbaum. "Mind games: Game engines as an architecture for intuitive physics". En: *Trends in cognitive sciences* 21.9 (2017), pp. 649-665.

[44] Pauli Virtanen et al. "SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python". En: *Nature Methods* 17 (2020), pp. 261-272. DOI: 10.1038/s41592-019-0686-2.

[45] Thomas Wisspeintner, Tijn Van Der Zant, Luca IocchiStefan Schiffer. "RoboCup@ Home: Scientific competition and benchmarking for domestic service robots". En: *Interaction Studies* 10.3 (2009), pp. 392-426.

[46] Jiajun Wu, Ilker Yildirim, Joseph J Lim, Bill FreemanJosh Tenenbaum. "Galileo: Perceiving physical object properties by integrating a physics engine with deep learning". En: *Advances in neural information processing systems* 28 (2015).

[47] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling ChangMatthias Grundmann. "Mediapipe hands: On-device real-time hand tracking". En: *arXiv preprint arXiv:2006.10214* (2020).

[48] Xuhui Zhou, Yue Zhang, Leyang CuiDandan Huang. "Evaluating commonsense in pre-trained language models". En: *Proceedings of the AAAI conference on artificial intelligence*. Vol. 34. 05. 2020, pp. 9733-9740.

ANEXOS

Anexo 1

Anexos

1.1. Parámetros utilizados en la experimentación

1.1.1. Detección de cubos

Para el filtrado en detección de posiciones se utiliza $\alpha = 0.25$.

Las nuevas posiciones son publicadas si estas superan el umbral de **4 mm** en posición o **0.1** en orientación según la ecuación 1.1.

$$orientation\_difference(q_1, q_2) = 1 - \langle q_1, q_2 \rangle^2 \tag{1.1}$$

1.1.2. Recalibración

Cuando se detecta un error promedio mayor a **15** (utilizando la ecuación 4.1) se realiza la recalibración.

El factor **K** es utilizado para aumentar el peso de la orientación en la función de distancia. Para todos los casos se utilizó $K = 1000$.

El proceso de minimización se detiene luego de **1000** iteraciones o si el error no disminuye más de $1e^{-}5$.

1.1.3. Agente de sorpresa

Cuando los cubos son detectados en una posición que supera los **15 mm** con respecto a la anterior, son marcados como sorprendentes.

1.2. Dominio de planificación

La definición del entorno se realizó en PDDL y es la siguiente:

Listing 1.1: Definición del entorno en PDDL.

```
(define (domain BLOCKS)
  (:requirements :strips :typing)
  (:types block)
  (:predicates  (on ?x - block ?y - block)
                (ontable ?x - block)
                  (clear ?x - block)
                  (handempty)
                  (holding ?x - block)
  )
  (:action pick-up
        :parameters (?x - block)
        :precondition (and (clear ?x) (ontable ?x) (handempty))
        :effect
                    (and (not (ontable ?x))
                         (not (clear ?x))
                         (not (handempty))
                         (holding ?x))
 )
  (:action put-down
              :parameters (?x - block)
              :precondition (holding ?x)
              :effect
                      (and (not (holding ?x))
                           (clear ?x)
                           (handempty)
                           (ontable ?x))
        )
  (:action stack
              :parameters (?x - block ?y - block)
              :precondition (and (holding ?x) (clear ?y))
```

```
            :effect
                              (and (not (holding ?x))
                                   (not (clear ?y))
                                   (clear ?x)
                                   (handempty)
                                   (on ?x ?y))
)
  (:action unstack
            :parameters (?x - block ?y - block)
            :precondition (and (on ?x ?y) (clear ?x) (handempty))
            :effect
                              (and (holding ?x)
                                   (clear ?y)
                                   (not (clear ?x))
                                   (not (handempty))
                                   (not (on ?x ?y)))))
```

1.3. Errores filtrados para la calibración

Durante la experimentación se observó que algunos de los cubos presentaban niveles de error sobre la media y en todos los casos. Se decidió, durante el análisis de estos resultados, descartar estas medidas.

1.3.1. Experimento según cantidad de cubos

En la figura A1 se puede observar que el cubo tres y el cubo cinco tienen en todos los casos un error muy sobre la media de los demás y sus intervalos de confianza también son muy extendidos. Debido a esto, se descartaron los cubos para la evaluación.

1.3.2. Experimento según posición de los cubos

En la figura A2 se puede observar que el cubo tres tiene en todos los casos un error muy sobre la media de los demás y su intervalo de confianza también es muy extendido. Debido a esto, se descartó el cubo para la evaluación.

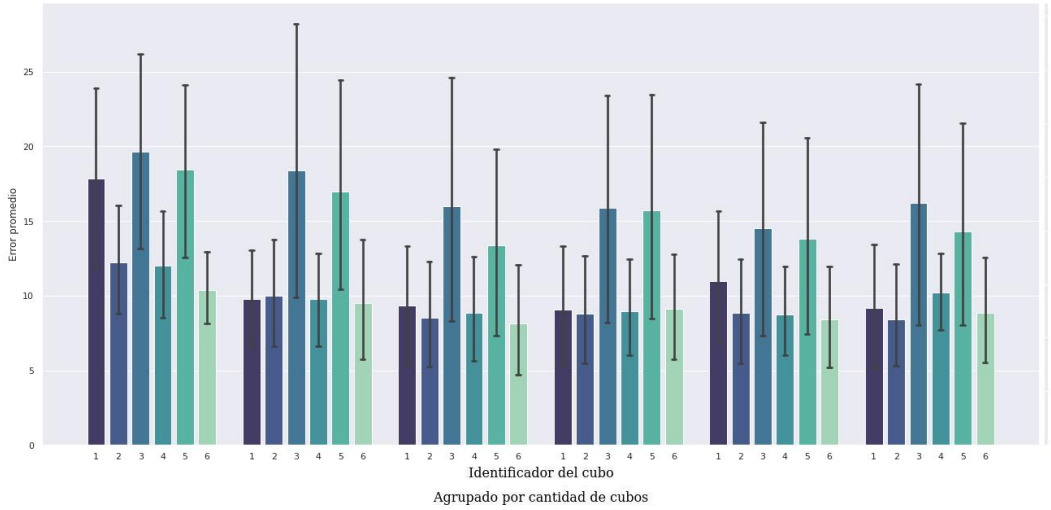


Figura A1: Medida de error para cada uno de los cubos y para cada cantidad de cubos utilizados para la calibración.

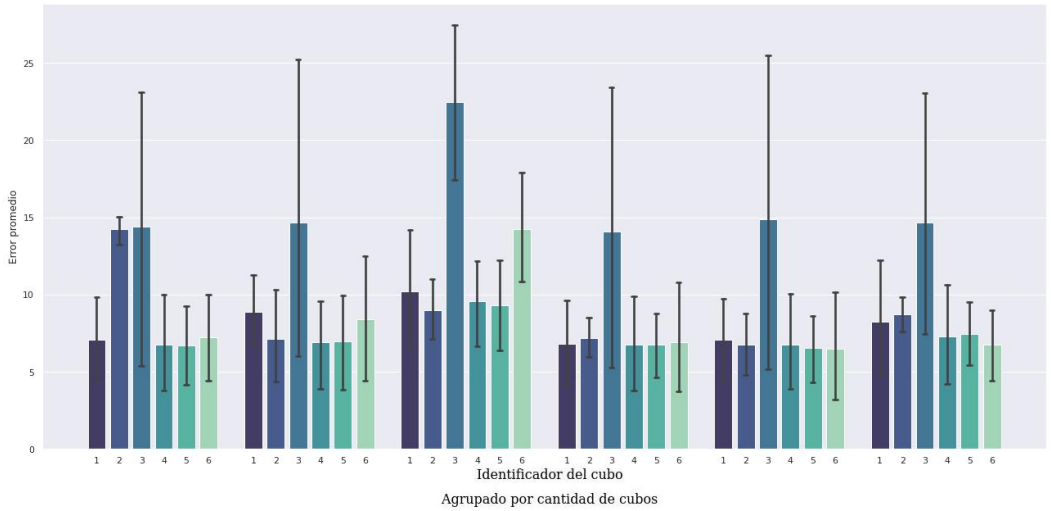


Figura A2: Medida de error para cada uno de los cubos y para cada configuración de cubos utilizada para la calibración.