
**UNIVERSIDAD CATOLICA DEL NORTE**

**FACULTAD DE INGENIERIA Y CIENCIAS GEOLOGICAS**

**DEPARTAMENTO DE INGENIERÍA DE SISTEMAS Y COMPUTACIÓN**

# Observatorios Robóticos: Utilización de software de robótica en la teleoperación de observatorios.

**Tutor**: José Gallardo Arancibia

**Alumno**: Andrés Villalobos Rivera

Reporte de Tesis

Magíster en Ingeniería Informática

Departamento de Ingeniería de Sistemas y Computación

Universidad Católica del Norte

Antofagasta, Chile, Noviembre de 2016

# Resumen:

La astroingeniería es la combinación de ingeniería y astronomía con objeto de resolver necesidades de la segunda con las técnicas más avanzadas que ofrece la primera. En los últimos 3 años el Gobierno de Chile ha encargado dos estudios a través de CONICYT, ("Astronomy, Technology, Industry: Roadmap For The Fostering Of Technology Development And Innovation In The Field Of Astronomy In Chile") y a través del Ministerio de Economía ("Estudio Capacidades y Oportunidades para la Industria y Academia en las actividades relacionadas o derivadas de la Astronomía y los grandes observatorios Astronómicos en Chile"), para analizar las perspectivas de desarrollo que tiene Chile en relación a la astronomía. Ambos estudios convergieron en la misma recomendación: Chile puede encontrar uno de sus principales nichos de desarrollo no en la astronomía como ciencia básica (donde de todos modos hay un futuro garantizado a nivel de usuarios) sino que en la astroingeniería, que permitirá a Chile explotar su cercanía física con estos proyectos. Estas recomendaciones se complementan con una ambiciosa propuesta a una década plazo, que comienza con la elaboración de un plan de desarrollo de la astronomía nacional, se sigue con un aumento de las inversiones estatales en esta área, y culmina con la creación de un par de centros de excelencia en astroingeniería, con la capacidad de desarrollar la tecnología que reemplazará a la tecnología actual en los observatorios del norte del país.

En esta tesis se aborda la problemática relacionada con el control y teleoperación de los Observatorios. Existe una gran cantidad de implementaciones de observatorios cada una de ellas con una opción o una solución diferente lo que llama bastante la atención, ¿Por qué cada observatorio posee una implementación diferente?. Para resolver estas interrogantes se hizo una extensiva revisión del estado del arte, llegando a la conclusión de que la comunidad al ser ten reducida, carece de las herramientas para desarrollar por las propias un software de control común y las opciones que existen son de pago y caras o bien no son lo suficientemente flexibles.

Como solución se propuso utilizar ROS para el control y posterior teleoperación de los observatorios, con el fin de dar a la comunidad astronómica, la oportunidad de ampliar las opciones de software de control para observatorios basados en una plataforma ampliamente utilizada en el mundo de la robótica tanto a nivel de investigación como industrial, proveyendo al mundo de la astronomía un framework con amplio soporte, trayectoria y que a la vez no limita a los profesionales que trabajan en esta área a un campo puntual, como lo sería un software especializado para astronomía.

El estudio concluye que ROS es una alternativa real para la implementación de observatorios remotos. Con la correcta implementación de driver apropiados (muchos de ellos no existían por que nadie había incursionado en esta área) el control de un telescopio y/o un observatorio es completamente viable y con el correcto desarrollo, el área de data handling a través de ROS es un paso natural.

# Abstract

# Índice de contenido

# Índice de ilustraciones

# CAPÍTULO I

## Introducción

*“La comunidad astronómica es reducida, distribuida y altamente especializada. [1]”*

---

*En este capitulo se realiza una breve introducción sobre el problema a ser enfrentado, la lejania de centros poblados en que deben ubicarse los observatorios y la importancia de la teleoperación. Se presentan los antecedentes técnicos y teóricos detrás del proyecto propuesto. Finalmente se presentan los objetivos y productos que es esperan lograr una vez finalizado este trabajo.*

# Introducción

Chile posee cerca del 70% de la capacidad de observación en el mundo [2], además posee 3 de los observatorios astronómicos mas grandes del mundo; Observatorio Cerro Paranal (VLT) [3], ALMA [4] y el proyecto Extremely Large Telescope (ELT) ELT[5] al que a fines del 2013 se otorgó a ESO la concesión del cerro Armazones [6] para su construcción.

Por otra parte, Chile esta potenciando de manera creciente la investigación y el desarrollo de la ciencia y la tecnología, de hecho, el presupuesto CONICYT entre los años 2009 – 2013 se ha visto incrementado en un 71,6%, mientras que entre 2006 y 2016 ha tenido un incremento aproximadamente del 377% según consta en las leyes de presupuesto de la nación [7][8], además dentro del presupuesto de CONICYT para el año 2012, las Ciencias Naturales y Exactas en conjunto con las Tecnologías, poseían cerca del 70% del presupuesto de CONICYT asignado a proyectos [9]. Esto significa que el gobierno de Chile tiene un claro interés en el desarrollo de la Ciencia y la Tecnología.

De la misma forma, CONICYT a través del fondo QUIMAL (Fondo Astronomía Quimal para el desarrollo de tecnologías para la Astronomía nacional) el año 2013, entregó por concepto de proyectos $615.546.331, en particular otorgó a la Universidad de Antofagasta (QUIMAL130004), un total de $138.000.000 que, en conjunto con CODELCO y la Universidad de Antofagasta, suman un monto total para el proyecto de $503.000.000 para la construcción de un Observatorio Astronómico, el primer observatorio Chileno.

# Observatorio

Un observatorio es un centro de investigación dedicado al estudio del cielo y está dotado de instrumentos para la observación de los fenómenos celestes.

El concepto de observatorio astronómico ha experimentado una profunda evolución con el pasar del tiempo. Antiguamente, cuando la astronomía estaba íntimamente ligada a las creencias religiosas, los observatorios coincidían con los templos destinados al culto de las divinidades. Es en la Edad Media cuando se afirma la concepción de observatorio como lugar de reunión de astrónomos e instrumentos. En los siglos sucesivos el observatorio se instala, por lo general, en una torre elevada de la ciudad.

Sin embargo, después de los primeros decenios del siglo XX se manifiesta la exigencia de alejarse de la contaminación química y luminosa de las metrópolis. Así se establecen los observatorios en lugares aislados y elevados, donde el cielo nocturno es oscuro y el número de días serenos cada año es muy elevado.

Desde el año 1996, gracias a los extraordinarios progresos de la física espacial y de las técnicas de exploración automática del espacio, se inició la construcción de los observatorios astronómicos orbitales.

Ahora es posible para un astrónomo aprovechar sus horas de observación con un gran instrumento,

sentado cómodamente en la habitación de su instituto universitario, controlando el telescopio a distancia a través de un terminal conectado a un computador central que realiza todas las funciones del gran instrumento [10].

El observatorio a ser construido por el proyecto QUIMAL, por requerimientos científicos estará ubicado lejos de la Universidad, por lo tanto el primer desafío a resolver es ¿como utilizar esta infraestructura?, ¿los estudiantes y científicos deberían viajar 2 horas en vehículo para llegar a este lugar y trabajar en el desierto?. La respuesta a estas interrogantes es lograr la completa automatización y robotización de este observatorio, con el fin de evitar trasladar personal y estudiantes a zonas lejanas, reduciendo costos, tanto de mantención como de uso de las instalaciones y finalmente generando información científica a menor costo.

Existen varios esfuerzos por lograr el control de un observatorio [11]–[19] pero muchos de ellos proponen sus propias soluciones, intentan desarrollar su propio software o utilizar uno ligado a la astronomía (que al ser profesionales son tan grandes y complejos, que resultan poco viables para un grupo pequeño de programadores [20]).

Sin embargo, y de acuerdo con [20] es preferible utilizar software libre, principalmente por que la comunidad astronómica es muy reducida y altamente especializada [1] por lo tanto se debe tratar de unir la mayor cantidad de esfuerzos en una solución para resolver este difícil problema. Cabe destacar que cada observatorio tiene sus propios requerimientos y necesidades, no es lo mismo contar con un observatorio de tipo Óptico que uno de Radiofrecuencias, ni tampoco es lo mismo tener un observatorio con continuas precipitaciones y variaciones climáticas que uno en medio del desierto donde el clima es mas estable, sin embargo comparten requerimientos funcionales similares que permiten generalizar la forma en que se pueden controlar.

En definitiva, un observatorio puede ser visto como un proyecto de domótica, donde se tiene un serie de sensores y actuadores, cada uno de ellos cumpliendo una función con alta precisión y siendo administrados por un sistema centralizado de control. Por esta razón se propone usar ROS, un framework de robótica ampliamente utilizado en el medio académico e industrial. Este framework entiende cada uno de sus componentes como un nodo computacional y se comunican entre ellos enviado y recibiendo mensajes, muy similar a como funcionan actualmente muchos observatorios, pero con la ventaja de ser un framework que viene del área de la robótica donde la comunidad es mas amplia y diversa.

## ROS

ROS (Robot Operating System) [21] es un framework flexible para el desarrollo de software para robots. Corresponde a una colección de herramientas, librerías y convenciones que buscan simplificar la creación de robustos y complejos comportamientos robóticos para una amplia variedad de robots.

ROS es un framework de computación distribuida, es decir posee equipos que se comportan como clientes y otros como servidores, lo que significa que cada robot es entendido como un computador y

sobre esto funciona el framework, usa la suposición de que todos sus nodos son un computador.

Finalmente, dado que los software en astronomía mas conocidos (VLTSW, ACS, RTS2 y INDI) están basados completamente en computación distribuida y difieren de ROS solo en la alta especialización astronómica, es completamente razonable pensar en utilizar ROS para el control de telescopios y/o observatorios. Por otra parte, software tan grandes como los antes mencionados resultan muy complicados de implementar y mantener cuando el equipo de trabajo es reducido y mas aún cuando no son especialistas en el framework. Para enfrentar este problema, se plantea la utilización del framework de robótica (ROS) para realizar el control de un observatorio ubicado en un lugar remoto. Como consecuencia, futuros investigadores no tendrían que invertir en desarrollar un software de astronomía, o tratar de adaptar alguno que ya existe, si no mas bien, tratar al observatorio como un robot y utilizar ROS para implementar el sistema de control y la teleoperación de esta infraestructura.

## Preguntas de Investigación

A raíz de la discusión presentada en los párrafos anteriores, se formularon las siguientes preguntas de investigación:

1. ¿Es posible utilizar o adaptar software de robótica en la automatización y control de telescopios y observatorios?
2. ¿Qué factores tienen en común ambos software (astronómico y de robótica) ?
3. ¿De que forma, el uso de tecnologías de libre acceso y ampliamente mantenidas podrían impactar en la implementación de nuevos observatorios?
4. ¿Contar con un software público para control de telescopio, hará más viable la investigación astronómica y construcción de nuevos centros de observación?
5. ¿Es ROS lo suficientemente robusto y confiable como para ser utilizado para el control de un observatorio profesional?

## Objetivos del Trabajo

El objetivo general de esta tesis de Magister consiste en Aplicar ROS para el control y la automatización de Telescopios y Observatorios y desarrollar los drivers necesarios para la interacción del telescopio con el framework. Para el correcto cumplimiento de este objetivo se han definido los siguientes objetivos específicos:

1. Analizar software de astronomía.
2. Estudiar y aplicar ROS.
3. Analizar compatibilidad de hardware con Linux y drivers asociados.
4. Diseñar drivers e integración con ROS.
5. Desarrollar Sistema SCADA.
6. Realizar la implementación en el observatorio.

## Aportaciones

En la siguiente sección, se declaran los aportes que se pueden obtener de este trabajo de tesis, en función de los objetivos mencionados.

- **Aportaciones teóricas y de investigación**: Como consecuencia del objetivo “Analizar software de astronomía” se realizará una extensiva revisión del estado del arte en cuanto a Software de control para Astronomía. En este análisis se revisarán tanto software de grandes observatorios como de pequeños observatorios, con el fin de tener una base de comparación para luego determinar puntos de intersección entre ROS y el software para astronomía.

- **Aportaciones Prácticas:** En este trabajo se explica cómo funciona un observatorio, que tipos de software pueden ser desarrollados para astronomía y cómo utilizar ROS en Astronomía.

## Estructura General

El informe de trabajo de tesis realizado, se ha organizado en 7 capítulos. A continuación se describe en forma resumida el contenido de cada uno de ellos.

En el capítulo 2, se presenta el marco teórico, en el cual se recorren diversas áreas de la robótica, las astronomía y la teleoperación, sentando las bases del trabajo de tesis. Más concretamente se estudian los siguientes tópicos:

- Conceptos generales de Astronomía y los equipamientos necesarios para llevar a cabo esta ciencia. Se estudian tópicos como Observatorios y Telescopios que son los principales sistemas.
- Descripciones generales sobre robótica, historia y terminología.
- Una completa descripción de qué es y cómo funciona ROS, framework propuesto en esta tesis para ser usado en Astronomía.

En el capítulo 3, se realiza un extenso estudio del estado del Arte en astronomía, los desafíos de ésta y como otros investigadores han resuelto los problemas que esta disciplina presenta. Entre los tópicos discutidos en este capítulo podemos destacar:

- Software Profesional y Amateur para el control de telescopios y Observatorios.
- Hardware a ser utilizado en el proyecto.

En el capítulo 4 se realiza la definición del problema, las hipótesis propuestas y los objetivos necesarios para comprobar la hipótesis planteada.

En el capítulo 5 se plantea la solución al problema planteado en el capítulo 4, en dicha solución se propone utilizar ROS para el desarrollo de sistemas de control para astronomía en pequeños observatorios puesto que la comunidad astronómica, es reducida, distribuida y altamente especializada, por lo tanto tener soporte de software en un framework específico de Astronomía es mucho más complicado que obtener el mismo soporte para un framework ampliamente utilizado en el ámbito de la robótica. Por otra parte, ROS posee todas las características necesarias para ser utilizado en Astronomía y así lo demostró la investigación realizada. Finalmente se logró realizar pruebas exitosas en el observatorio Ckoirama de la Universidad de Antofagasta utilizando este concepto.

En el capítulo 6 se realiza una discusión de los resultados obtenidos, destacando entre ellos las pruebas exitosas de este concepto en el Observatorio Ckoirama de la Universidad de Antofagasta. Además se definieron una serie de métricas que dan cuenta de las ventajas de utilizar software de Robótica en Astronomía, las ventajas de utilizar frameworks especializados en computación distribuida y los problemas encontrados durante la implementación de este proyecto.

En el capítulo 7 se enuncian y discuten las principales aportaciones del trabajo de tesis en el contexto de su aplicación para la automatización, control y teleoperación de observatorios robóticos. Por último y a continuación, se propone un conjunto de líneas futuras de investigación en la temática tratada.

# CAPÍTULO II

## Marco Teórico

---

*En este capitulo se presentan los aspectos teóricos que sustentan el desarrollo de esta tesis de Magister. Para comenzar, se realiza una introducción a la Astronomía, en qué consiste y cual es su foco de investigación. Para la investigación astronómica, es de vital importancia contar con un Observatorio, estos recintos son los centros de investigación en donde se ubican los instrumentos para medir y observar el cielo, usualmente se ubican distantes de centros urbanos con el fin de evitar la contaminación lumínica o de radiofrecuencia.*

*Luego se describe en que consiste la robótica y porque se propone utilizar robótica en astronomía, para luego explicar lo que es un framework, puesto que ROS (tratado en la sección siguiente) es un framework. Finalmente se hace una descripción de que es ROS y cómo funciona para tener las bases suficientes para analizar si ROS es efectivamente utilizable en astronomía y cuales serían sus limitaciones.*

# Marco Teórico

## Astronomía

La astronomía no es solo una ciencia, si no que también una parte importante de la cultura, por lo tanto, es natural (al igual que en muchos otros países) que la astronomía tenga un rol importante desde la prehistoria. Fue cultivada tanto por la iglesia en monasterios y escuelas así como también en las cortes de la nobleza en el reino Checo medieval. Se puede definir como astronomía, a la ciencia que investiga toda la materia y la energía en el universo, lo que implica el estudio de su distribución, composición, estados físicos, movimientos y evolución [22].

La astronomía es dividida en 2 ramas principales distinguidas como Astrometría y Astrofísica [23]; la primera relacionada con la determinación de los lugares de investigación de los cuerpos celestiales y la segunda relacionada con la investigación de sus naturaleza química y física. Sin embargo esta división es relativamente nueva. Las posibilidades de la ciencia antigua se limitaban a fijar la posición aparente de los objetos en una esfera. Tampoco se hizo ningún intento por razonar acerca de los hechos observados sino hasta que los Griegos construyeran un sistema especulativo, el cual fue finalmente desplazado por el amplio espectro de la teoría de la Gravedad. Por otro lado la Astronomía descriptiva tuvo su impulso con la invención del telescopio, y las instalaciones que se les brindo para la estrecha vigilancia de los habitantes del cielo; mientras la astronomía práctica se mejoro continuamente con el refinamiento y las mejoras de las artes ópticas y mecánicas. Actualmente, se podría decir que la astrofísica a absorbido la astronomía descriptiva, y la astrometría necesariamente incluye la investigación práctica. La astronomía matemática, basada en la ley de la gravitación mantiene su lado aparte, aunque todo depende de la perfección de sus teorías, la ampliación del ámbito de aplicación a lo largo del tiempo y las exploraciones en nuevas direcciones.

### Astronomía Descriptiva

Se podría decir que la astronomía descriptiva tuvo su origen con la invención del telescopio, atribuido a Hans Lippershey en 1608. Su aplicación en el estudio de los cuerpos celestiales por Galileo Galilei y otros, condujo a una multitud de descubrimientos asombrosos. Los satélites de Júpiter, las fases de Venus, las montañas de la Luna, las manchas solares, los anillos de Saturno, todos descritos con un pequeño instrumento parecido a un lente monocular para operar, contribuyendo cada uno a su manera, una revelación significante y sorprendente. Con ello también se cambio la percepción de la composición estelar de la Vía Láctea, representando así el primer paso hacia la exploración sideral.

Johan Kepler (1571-1630) inventó en 1611 junto con el padre Scheiner de Ingolstadt (1575 – 1650) quien fuera su primer empleado, el primer telescopio de refracción y los descubrimientos del padre Scheiner estuvieron muy ligados al desarrollo del potencial de este telescopio. Christian Huygens (1629 – 1695) resolvió en 1665 el misterio de las “ansae” de saturno, que resultaron ser los anillos, divididos en 2 por Giovanni Domenico Cassini (1625 – 1712)  en 1675.

Titan, la mas grande luna de Saturno, fue detectada por Huygens en 1655 y cuatro lunas mas para el año 1684. La nebulosa de Andromeda fue notada por Simon Marius en 1612, la nebulosa de Orión por J.B. Cysatus, un jesuita suizo en 1618, además fueron reconocidas varias estrellas variables y múltiples.

## Astronomía Teórica

La astronomía teórica, sin embargo, fue lejos compensada por los logros prácticos del siglo XVII. Kepler publica 2 de las "3 Leyes" en 1609 y la tercera en 1619. Lo importante de estas grandes generalizaciones son:

- Que los planetas describen elipses en las cuales el sol ocupa su único foco.
- Que la linea recta que une cada planeta con el sol (el vector radio) recorre áreas iguales en tiempos iguales.
- Que los cuadrados de los periodos planetarios son proporcionales al cubo de la distancia media desde el Sol.

El plan geométrico de movimiento en el sistema solar, fue establecido por una intuición maravillosa pero fue reservado por Sir Isaac Newton (1634 – 1727) para exponer su insignificancia demostrando que la misma fuerza que actúa de forma uniforme, regula las revoluciones celestiales y obliga a los cuerpos pesados a caer hacia la superficie de la tierra. La ley de la gravedad, publicada en 1687 en "Philosophiae Naturalis Principia Mathamatica", tiene el siguiente efecto: cada partícula de materia se atrae entre si con una fuerza directamente proporcional a sus masas e inversamente proporcional al cuadrados de la distancia que los separa. Su validez fue probada comparando la cantidad de desviación orbital de la luna en un segundo con la velocidad a la que una manzana cae sobre un huerto. Al tener en cuenta la distancia a la Luna, las dos velocidades probaron ser totalmente perfectas, la identidad de la gravedad terrestre con la fuerza controlando las revoluciones de un cuerpo celestial, pero esto fue solo el comienzo, el colosal trabajo lograba calcular las consecuencias de la ley en los mas mínimos detalles, comparándolos con el cielo.

## Astrofísica

El principio fundamental del análisis de espectro, enunciado por Gustav Kirchhoff (1824 – 1887) depende de la equivalencia entre la emisión y la absorción. Esto significa que, si una luz blanca es transmitida a través de vapores brillantes, retienen solo aquellas pequeñas secciones con las que pueden brillar ellos mismos. Si la fuente de luz blanca es mas caliente que el vapor que las retiene, el resultado es un espectro prismático, interrumpido por lineas negras, característico de la naturaleza química de la sustancia que lo origina. Este es exactamente el caso del sol y las estrellas. La radiación blanca que emana de la fotosfera es encontrada, cuando es dispersada en un espectro, que luego es atravesado por varios rayos oscuros que indican la absorción por estrato gaseoso, la composición es determinada usando el principio de Kirchhoff.

El mismo Kirchhoff, identifico en 1861 como prominentes componentes solares; sodio, hierro,

magnesio y calcio, el cromo fue identificado por A.J. Angström (1814 – 1874), el Helio fue descompuesto por Sir Norman Lockyer en 1868 y alrededor de 40 sustancias elementales son conocidas con una certeza aproximada de ser comunes tanto en la tierra como en el sol. La química de las estrellas es estrictamente análoga a la del Sol, aunque su espectro exhibe diversidades sintomáticas de una considerable variedad de estados físicos.

El padre Angelo Secchi, S.J., (1818 – 1878) basado en esta diversidad en 1863-1867 realizó una clasificación de estrellas en 4 órdenes, siendo aun considerado como fundamental y complementado por el doctor Vogel en 1874, con una interpretación evolutiva en función de los diferentes tipos de espectro, estas están asociadas a variados estados de progreso desde uno tenue y rudimentario hacia una condición compacta. Desde 1879, cuando Sir William Huggins aseguró impresiones de un rango extendido de luz ultravioleta de una estrella blanca, el espectro estelar a sido investigado mayormente a través de fotografías, los resultados además de ser precisos son permanentes y mucho mas completos que aquellos que se obtienen a través de la simple vista. Sir William además descubrió en 1864, los espectros de linea brillante de un cierto tipo de nebulosa, con la cual se pudo determinar su composición gaseosa y reconocida como de origen de carbono, la banda típica de colores del espectro de un cometa notado 4 años antes, aunque sin una identificación específica por G.B. Donati (1827 – 1873) en Florencia.

El principio de Doppler, el cual indica que la luz altera su refrangibilidad hacia el final del movimiento de la fuente de emisión, fue aprobada por primera vez para ser usada en las investigaciones astronómicas en 1868. El criterio de la velocidad, ya sea de la recesión o el enfoque, es producido por el desplazamiento de las líneas espectrales de los lugares estándar, el método alcanzo un alto grado de precisión con la adaptación del Dr. Vogel en 1888. Desde entonces, ha demostrado ser un método bastante fructífero. Su trabajo permitió al Dr. Vogel demostrar la realidad de los eclipses de Algol, mostrando que la estrella giraba en turno a un compañero oscuro en el periodo idéntico de cambio de luz, estos son los primeros descubrimientos de binarias espectroscópicas no eclipsantes y fueron realizadas en el Harvard College en 1889. Estos interesantes sistemas no pueden ser claramente distinguidos desde una estrella doble telescópica, la cuales de hecho se cree que tienen influencia en las mareas, sus periodos varían entre un unas pocas horas y varios meses, además sus componentes son tan diferentes en términos de luminosidad que solo uno de sus componentes, deja un rastro en la placa sensible. El número conocido de este tipo de estrellas ha ido creciendo y podría seguir creciendo indefinidamente.

La fotografía con luz de día de las prominencias solares fue intentada en 1870 por el profesor Young de la universidad de Princeton y el material generado fue revisado por el Dr. Braun en 1872. No hubo ningún éxito con estos experimentos si no hasta 1891, cuando Hale de Chicago y M. Deslandres en París, cada uno por separado, construyeron imágenes de esos objetos fuera de los rayos de calcio en la luz dispersa, filtrada a través de una doble rendija móvil en las placas de fotografía. El invento del profesor Hale, el espectroheliógrafo le permitió además delinear el disco del sol en cualquiera de las luces seleccionadas, que permitió determinar la existencia de grandes masas de calcio y flóculos de

hidrógeno apilados en diferentes alturas sobre la superficie solar. [23]

En los párrafos anteriores, se describió a grandes rasgos los principales descubrimientos en astronomía y los distintos tipos de investigaciones que se siguen en el ámbito de la astronomía, sin embargo, sin el desarrollo de instrumentos que permiten ver la luz mas allá de lo visible para el ojo humano, que permiten analizar mucho mas información que el ojo humano y que además permite guardar la información para su posterior análisis, estos avances no hubiesen sido posibles. Es por esta razón que el desarrollo de instrumentos y mejor infraestructura para el desarrollo de la astronomía, es algo fundamental. Esto también explica el interés por desarrollar nuevas tecnologías y construir nuevos centros donde alojar estos instrumentos, dichos centros son conocidos como Observatorios, en los cuales se encuentran alojados los instrumentos que permiten capturar y analizar la luz y los telescopios que permiten dirigir y acumular la luz del cielo para luego dirigirla a los instrumentos.

El apoyo de la ingeniería en el desarrollo de la ciencia ha sido fundamental, puesto que ha permitido el desarrollo de sofisticados instrumentos capaces de lograr recuperar valiosa información desde el cielo. Sin embargo, las condiciones para que estos observatorios sean ubicados, son agrestes por tanto la necesidad de robotizar estos centros para que puedan ser teleoperados, es uno de los desafíos del siglo XXI.

## Observatorios

Cuando la gente piensa en la observación astronómica, sin siquiera pensar en ello, usualmente piensan en un edificio con un gran domo que en su interior posee un telescopio [24]. Y sí, éste es un tipo de observatorio. Sin embargo un observatorio astronómico, corresponde a cualquier lugar donde se realizan observaciones de los fenómenos celestiales. Este podría ser un campo abierto, una carpa, un jardín o cualquier tipo de construcción puesta al servicio de este propósito. Los antiguos observatorios tendían a ser torres, tejado o lugares altos, plataformas abiertas, pero hay pocas fuentes como para contarnos exactamente como se veían o como estas fueron usadas.

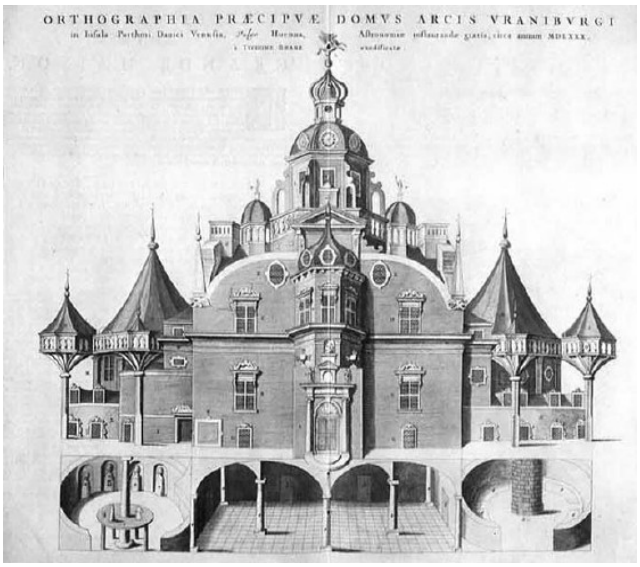



*Ilustración 1: El lujoso observatorio de Tycho Brahe, fuente [90]*

La ilustración 1, muestra el observatorio más grande de la era pre-telescopio, Tycho Brahe. Este fue un lugar que fue construido por el mismo Tycho en la isla de Hven, en el Báltico, al cual bautizó como Uraniborg o “castillo de los cielos. La observación pareciera haber sido realizada desde los balcones abiertos y los miradores cubiertos. En términos generales, en la era pre-telescópica, los instrumentos no eran protegidos del clima, aunque Brahe construyó su segundo observatorio por debajo del nivel del suelo para protegerlo del viento del Báltico. Las construcciones de observatorios siguieron siendo “raras”. Galileo hizo sus mas grandes descubrimientos desde el jardín y desde las ventanas de su casa.

Incluso ya bien entrada la era del telescopio, las construcciones de observatorios específicamente diseñados eran poco comunes y los observatorios con domos eran desconocidos. El diseño de Wren para el Observatorio Real de Greenwich, fue uno de los primeros intentos de resolver a priori los problemas arquitectónicos para acomodar los telescopios en el edificio y lograr protegerlos y además entregar un acceso intensivo a la observación cielo. Esta sigue siendo una solución elegante, con sus altas ventanas y grandes espacios bien adecuados a los instrumentos refractarios de largo foco e instrumentos de tránsito diario. Mas tarde los grandes observatorios de la temprana era telescópica, eran con frecuencia cerrados completamente como el de Earl de Rosse en Birr, Irlanda, con su enorme telescopio (por 72 años el mas grande del mundo) suspendido entre dos muros masivos, o el sitio del totalmente impráctico telescopio de Craig, en su tiempo el telescopio de refracción mas largo del mundo, peligrosamente colgado desde una torre construida con este propósito en Wandsworth Common en Londres.

Los observatorios con domos parten a mediados del siglo XIX. El primer observatorio con domo en Inglaterra, pareciera ser el domo con el telescopio ecuatorial Northumberland de la universidad de Cambridge, diseñado por Airy en 1835. El diseño de los domo giratorios con una abertura que se puede

abrir desde la base a la cima, es un tipo de diseño óptimo para un observatorio astronómico, el cual provee la mas completa protección posible para el equipamiento, además de permitir un completo acceso al cielo. Esta es la razón por la cual, han sido generalmente la solución favorita de los profesionales de las astronomía, por restricciones de presupuesto, y dificultad de ingeniería, es mas probable que sean menos tomados en consideración que el promedio de los observatorios amateurs. La forma clásica de un domo es parecida a la esfera celestial, sin embargo, el domo no tiene que tener una sección transversal circular y algunos recientes ejemplos no necesariamente tienen la forma clásica, han sido simplificadas a una forma cilíndrica mas simple o incluso una forma de cubo como el observatorio con el telescopio de mayor resolución del mundo, el “Large Binocular Telescope”.

### Tipos de observatorios

Los observatorios pueden ser ópticos (foco de esta investigación) y también del tipo radiofrecuencia, que miden la luz a través del uso de instrumentos que analizan ondas de radio. En el caso de los observatorios ópticos, el instrumento mas utilizado es el telescopio, que es el que focaliza la luz en un punto para luego ser capturada u observada, en el caso de un telescopio amateur, la luz podría simplemente ser capturada por el ojo humano, pero con el avance de la tecnología, actualmente es posible añadir instrumentos de captura de imágenes a estos telescopios, con lo cual se puede capturar una imagen para posteriores análisis, el desarrollo de la tecnología a la hora de capturar imágenes ha crecido enormemente, y la posibilidad de guardar estas observaciones ha posibilitado el análisis mas cuidadoso de estas observaciones.

Por otro lado se encuentran los observatorios de radiofrecuencia cuyo principal objetivo es analizar la luz como una onda radiación electromagnética, con lo cual se analizan las intensidades de energía que provienen de la atmósfera y en función de esta energía se, construyen las imágenes o se analizan los fenómenos estelares.

## Telescopio

Se denomina telescopio (del griego 'lejos' y 'observar') al instrumento óptico que permite ver objetos lejanos con mucho más detalle que a simple vista al captar radiación electromagnética, tal como la luz.

Gracias al telescopio el ser humano pudo por fin, empezar a conocer la verdadera naturaleza de los cuerpos celestes que nos rodean y nuestra ubicación en el universo. El primer uso reportado, fue el de Galileo Galilei en 161, quien lo usó para mirar la Luna, el planeta Júpiter y las estrellas.

El foco de esta investigación es lograr el control de un observatorio astronómico de tipo óptico, aunque no se descarta que sea aplicable a antenas para observatorios radioastronómicos. En el mercado es posible actualmente encontrar diversos tipos de telescopios con diferentes monturas.

### Tipos de Telescopios

El parámetro más importante de un telescopio es el diámetro de su “lente objetivo”. Un telescopio de aficionado, generalmente tiene entre 76 y 150 mm de diámetro y permite observar algunos detalles

planetarios y muchísimos objetos del cielo profundo (cúmulos, nebulosas y algunas galaxias). Los telescopios que superan los 200 mm de diámetro permiten ver detalles lunares finos, detalles planetarios importantes y una gran cantidad de cúmulos, nebulosas y galaxias brillantes.

Para caracterizar un telescopio y utilizarlo, se emplean una serie de parámetros y accesorios:

- **Distancia focal**: es la longitud focal del telescopio, que se define como la distancia desde el espejo o la lente principal hasta el foco o punto donde se sitúa el ocular.
- **Diámetro del objetivo**: diámetro del espejo o lente primaria del telescopio.
- **Ocular**: accesorio pequeño que colocado en el foco del telescopio, permite magnificar la imagen de los objetos.
- **Lente de Barlow**: lente que generalmente duplica o triplica los aumentos del ocular cuando se observan los astros.
- **Filtro**: pequeño accesorio que generalmente opaca la imagen del astro pero que dependiendo de su color y material permite mejorar la observación. Se ubica delante del ocular, y los más usados son el lunar (verde-azulado, mejora el contraste en la observación de nuestro satélite), y el solar, con gran poder de absorción de la luz del Sol para no lesionar la retina del ojo.
- **Razón Focal (f/ratio)**: es el cociente entre la distancia focal (mm) y el diámetro (mm).
- **Magnitud límite**: es la magnitud máxima que teóricamente puede observarse con un telescopio dado, en condiciones de observación ideales. La fórmula para su cálculo es: m(límite) = 6,8 + 5log(D); siendo D el diámetro en centímetros de la lente o el espejo del telescopio.
- **Aumentos:** es la cantidad de veces que un instrumento multiplica el tamaño aparente de los objetos observados. Equivale a la relación entre la longitud focal del telescopio y la longitud focal del ocular (DF/df). Por ejemplo, un telescopio de 1000 mm de distancia focal, con un ocular de 10mm de df. proporcionará un aumento de 100 (se expresa también como 100X).
- **Trípode:** conjunto de tres patas generalmente metálicas que le dan soporte y estabilidad al telescopio.
- **Portaocular:** orificio donde se colocan el ocular, reductores o multiplicadores de focal (p.ej lentes de Barlow) o fotográficas.

Cada uno de estos elementos pueden ser utilizados en diferentes tipos de telescopios:

**Reflector:**

Un telescopio reflector es un telescopio óptico que utiliza espejos en lugar de lentes para enfocar la luz y formar imágenes. Los telescopios reflectores o Newtonianos utilizan dos espejos, el primero en el extremo del tubo (espejo primario), que refleja la luz y la envía al segundo (espejo secundario) quien

luego envía la luz al ocular. En la ilustración 2 se puede apreciar el esquema de funcionamiento de un telescopio reflector.

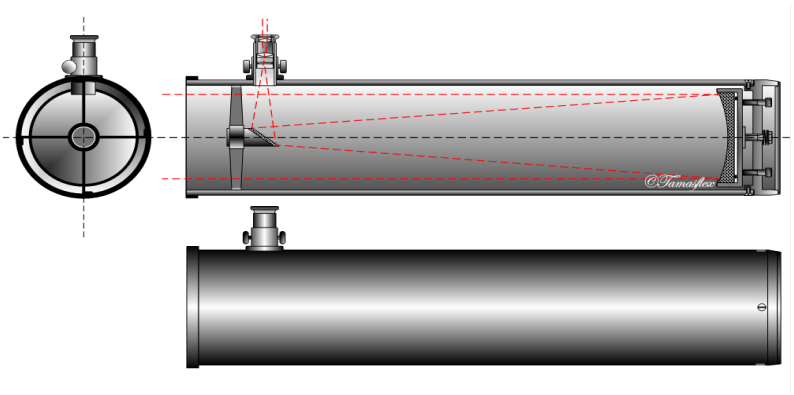


*Ilustración 2: Diagrama de Telescopio Reflector*

Un telescopio refractor es un sistema óptico centrado, que capta imágenes de objetos lejanos utilizando un sistema de lentes convergentes en los que la luz se refracta. La refracción de la luz en la lente del objetivo, hace que los rayos paralelos procedentes de un objeto muy alejado (en el infinito), converjan sobre un punto del plano focal. Esto permite mostrar los objetos lejanos mayores y más brillantes , en la ilustración 3 se puede apreciar un esquema de un telescopio refractor.

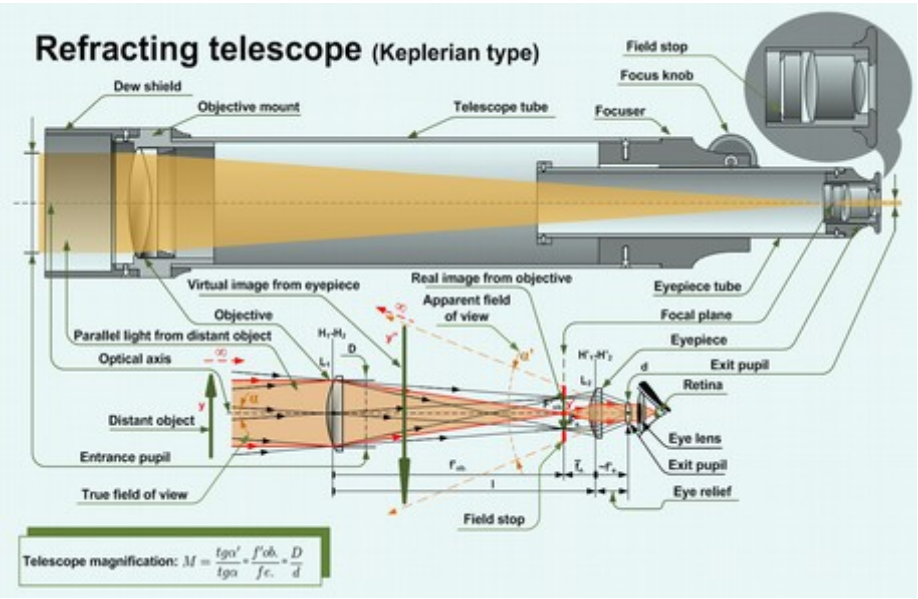



*Ilustración 3: Diagrama de un telescopio refractor tipo Kepler*

El Cassegrain es un tipo de telescopio reflector que utiliza tres espejos. El principal es el que se encuentra en la parte posterior del cuerpo del mismo. Generalmente posee forma cóncava paraboloidal, ya que ese espejo debe concentrar toda la luz que recoge en un punto que se denomina foco. La distancia focal puede ser mucho mayor que el largo total del telescopio.

El segundo espejo es convexo se encuentra en la parte delantera del telescopio, tiene forma hiperbólica y se encarga de reflejar nuevamente la imagen hacia el espejo principal, que se refleja (en su versión original) en otro espejo plano inclinado a 45 grados, enviando la luz hacia la parte superior del tubo, donde está montado el objetivo.

En otras versiones modificadas el tercer espejo, está detrás del espejo principal, en el cual hay un orificio central por donde la luz pasa. El foco en este caso, se encuentra en el exterior de la cámara formada por ambos espejos, en la parte posterior del cuerpo. En la ilustración 4 se puede apreciar un diagrama de un telescopio tipo Cassegrain.

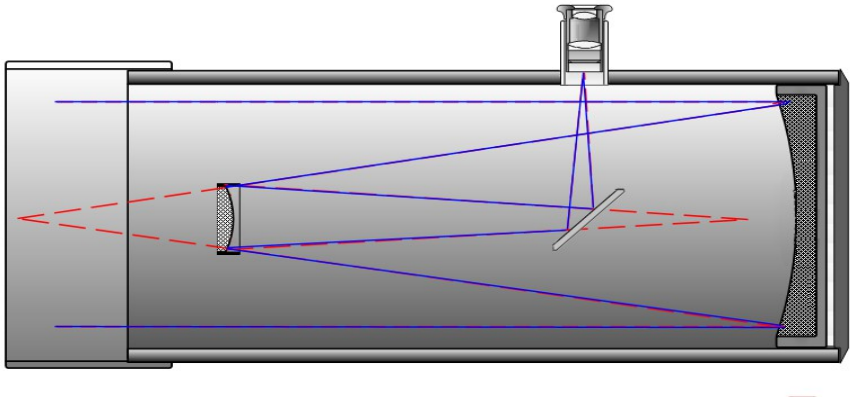


*Ilustración 4: Diagrama de telescopio tipo Cassegrain*

**Montura altazimutal:**

Una montura de telescopio sencilla es la montura altitud-azimut o altazimutal (ver ilustración 5). Es similar a la de un teodolito. Una parte gira en azimut (en el plano horizontal), y otro eje sobre esta parte giratoria permite además variar la inclinación del telescopio para cambiar la altitud (en el plano vertical). Una montura Dobson es un tipo de montura altazimutal que es muy popular dado que resulta sencilla y barata de construir.

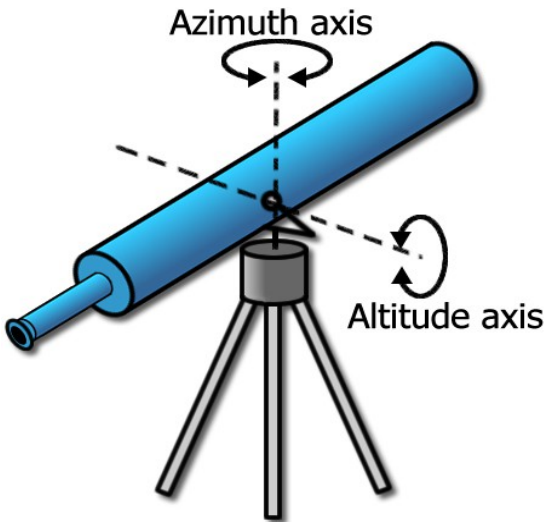



*Ilustración 5: Diagrama montura altazimutal*

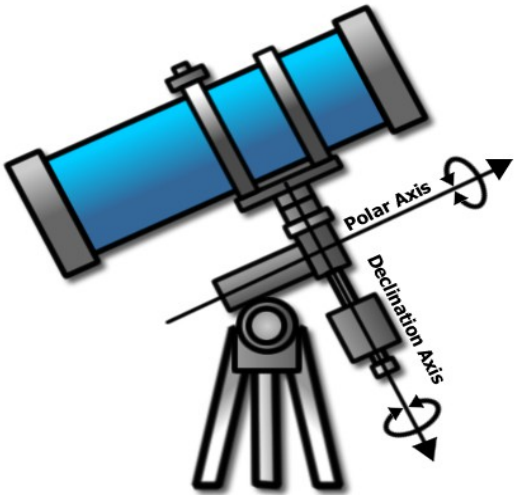



*Ilustración 6: Diagrama montura ecuatorial*

El principal problema de usar una montura altazimutal, es que ambos ejes tienen que ajustarse continuamente para compensar la rotación de la Tierra, incluso haciendo ello controlado por un computador, la imagen gira a una tasa que varía dependiendo del ángulo de la estrella con el polo celeste (declinación). Este efecto (conocido como rotación de campo) hace que una montura altazimutal resulte poco práctica para realizar fotografías de larga exposición con pequeños telescopios.

La mejor solución para telescopios astronómicos pequeños consiste en inclinar la montura altazimutal de forma que el eje de azimut resulte paralelo al eje de rotación de la Tierra; a esta se la denomina una montura ecuatorial (ver ilustración 6).

Existen varios tipos de montura ecuatorial, entre los que se pueden destacar la alemana y la de horquilla.

**Otras monturas**

Los grandes telescopios modernos usan monturas altazimutales controladas por computador para exposiciones de larga duración, o bien hacen girar los instrumentos, o tienen rotadores de imagen de tasa variable en una imagen de la pupila del telescopio.

Hay monturas incluso más sencillas que la altazimutal, generalmente para instrumentos especializados. Algunos son: de tránsito meridiano (sólo altitud); fijo con un espejo plano móvil para la observación solar. Sin embargo, debido a que el foco de la investigación son las monturas altazimutal, describir el resto de monturas no es necesario.

## Robótica

Los robots se han ido incorporando en la vida diaria en la última mitad del siglo, lo que alguna vez fue solo ciencia ficción ahora poco a poco se va convirtiendo en realidad. Actualmente, todas las personas que viven en un mundo desarrollado, se benefician de los avances de la robótica. A pesar de que los robots no han sido incorporados en la vida diaria como se describe en algunas series y/o películas (Star Wars, I Robot, Blade Runner y Artificial Inteligence por nombrar algunas) ésto si se ha desarrollado con creces en las industrias de manufactura, servicios y medicina, todas ellas han incorporado la robótica para ayudar a mejorar la eficiencia y la precisión de los procesos y tareas a realizar. Los robots nos ayudan a construir nuestras máquinas, empacan nuestra comida y lavan nuestros autos [25], en las ilustraciones 7, 8 y 9 se pueden apreciar algunas de estas maravillosas máquinas.

*Ilustración 7: Robots Kuka en manufactura de vehículos*

*Ilustración 8: Robots paletizadores*

*Ilustración 9: Robot de cirugia teleoperado*

Las primeras máquinas humanoides fueron mencionadas por los antiguos Griegos, descritos por Al-Jazari en el siglo XIII y luego diseñadas por Leonardo da Vinci en el siglo XV. Las máquinas parecidas a los humanos o las máquinas parecidas a los animales fueron una moda en Europa en el siglo XVIII con la creación del flautista, el músico, el dibujante y el pato que digiere [26] (ver ilustraciones 12, 13, 11 y 10 respectivamente).

*Ilustración 12: El Flautista*

*Ilustración 13: El músico*

*Ilustración 11: El dibujante*

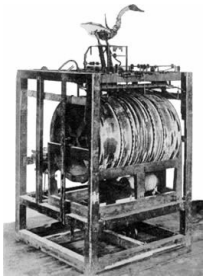


*Ilustración 10: El pato que digiere*

El término robot fue usado por primera vez por el dramaturgo checoslovaco Karel Capek (1890 – 1938) en 1920 en su obra R.U.R (Rossum's Universal Robots). La obra se sitúa en una remota isla en el medio del océano, en una instalación de producción de robots que están siendo vendidos como mano de obra barata a todo el mundo. La esposa del director de la fábrica usa sus encantos femeninos para convencer al ingeniero de producción, Dr. Gall, de agregar un alma a los robots. Gall es convencido y al nuevo despertar, los robots rápidamente se dan cuenta de su superioridad tanto mental como física respecto de la raza humana, por lo que se dedicaron a aniquilar a la humanidad. Este es un tema recurrente en las obras de ciencia ficción del siglo XX.

La palabra en si está derivada del sustantivo checo “robota” que significa “trabajo pesado”, “servidumbre” o “trabajo”, un *robotnik* es la palabra checa para denominar a un “obrero” o “campesino”.

Está claro que los robots de Capek no se parecían en nada a la fusión de metal, plástico y circuitería como los robots de hoy en día, aunque pudieron parecer así en sus producciones. Cuando se describen las máquinas y mecanismos presentes en la fábrica de producción de robots, Capek usa términos como bateas, tambores y molinos para crear el cuerpo de los Robots. Con esto nos podemos dar una idea de la visión de Capek, básicamente estaba basada en elementos biológicos, lo cual tiene mucho sentido, considerando desde un punto de vista práctico, los actores humanos tendrían que luego representar a estos personajes en el escenario. Lo que la creación de Capek's nos brinda no es una comprensión moderna de los robots, si no el concepto de obrero artificial, la cual es muy útil.

El escritor de ciencia ficción Isaac Asimov (1920 – 1992) es tal vez el mayor responsable por como percibimos a los robots en nuestros tiempos. En su novela corta “Runaround” en 1942, acuñó el término robótica, cuyo significado sería el campo de la ciencia dedicado a la construcción y estudio de robots. Pero aun mas importante, en dicha obra es la primera vez donde se observan las “Leyes de la Robótica”: [27]

1. Un robot no puede dañar a un ser humano o, por no realizar alguna acción permitir que un ser humano resulte dañado.
2. Un robot debe obedecer las ordenes dadas por un ser humano excepto cuando estas ordenes estén en conflicto con la primera ley.
3. Un robot debe proteger su propia existencia tanto como su protección no esté en conflicto con la primer y segunda ley.

Asimov concibe las Leyes de la Robótica para imponer orden a la libertad de la que gozaban sus robots ficticios. Desde entonces los robots han sido ampliamente descritos en la literatura y cine de ficción como amigo o enemigo del hombre.

Cerca de la última mitad el siglo XX, los robots similares a los de Capek se han convertido en una realidad y son usados para realizar trabajos mecánicos en fábricas, en un esfuerzo por minimizar el error humano y evitar daños, así como también ayudando a incrementar la producción y la eficiencia.

William Grey Walter, un investigador en cibernética, construye su primer robot móvil en 1948, el cual era capaz de buscar una fuente de luz y una estación de recarga. El movimiento de la Inteligencia Artificial (AI) fue fundado en la conferencia de Dartmouth en 1956 y dio lugar a una nueva investigación en robótica en las universidades de los Estados Unidos, Europa y Japón. [26]

La transición desde la ciencia ficción a la realidad se remonta al trabajo de Ray Goertz en el Laboratorio Nacional de Argonne hacia finales de 1940, quien desarrollo un brazo manipulador controlado remotamente para el manejo de materiales radioactivos. Luego en 1954, George Devol patenta el Unimation (Universal Automation), un brazo que utiliza coordenadas polares en una pista recta y que era controlado por señales magnéticas en un tambor giratorio. La máquina era programable, por lo tanto constituye un prototipo de los robots industriales modernos. Luego, Devol conoció a Engelberger y en conjunto fundaron Unimation en 1956, que rápidamente se convirtió en Unimation Inc. Una división de una consolidada corporación de Diesel (CONDEC). Tiempo después en 1960, vendieron su primer robot a General Motors para asistir en la producción de automóviles.

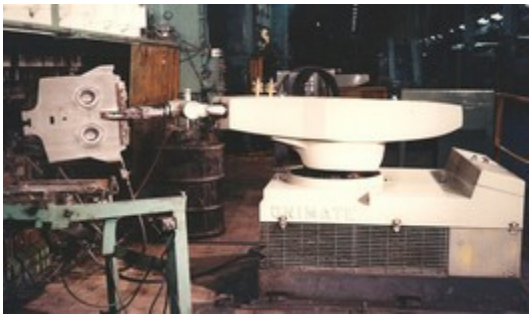


*Ilustración 14: Unimate, primer robot manipulador de la empresa Unimation*

Desde el primer uso del Unimate (ver ilustración 14) en la linea de producción en 1961, la aplicación de la robótica en la industria ha crecido enormemente. Los robots han sido utilizados en una variedad de aplicaciones incluyendo la investigación del océano profundo (ver ilustración 17), la exploración espacial (ver ilustración 15), el uso militar y para realizar la búsqueda en misiones de rescate (ver ilustración 16) [25][26].

*Ilustración 15: Zöe, prototipo de robot explorador de la Universidad Carnegie Mellon*

*Ilustración 17: Robot explorador submarino*

*Ilustración 16: Robot cuadrúpedo prototipo de DARPA*

En todos los casos, la robótica tiene como objetivo duplicar o mejorar las funciones del ser humano o sirviendo en situaciones muy peligrosas para el trabajo directo de un humano.

Existe una variedad de clasificaciones para los diferentes tipos de robots. Los robots pueden ser caracterizados como brazos automáticos, dispositivos móviles, rotadores o como dispositivos teleoperados. Adicionalmente, estos pueden ser activos, semiactivos o pasivos. Los dispositivos activos son totalmente programables y llevan a cabo tareas de forma independiente. Uno podría imaginar a un médico entrando en una tomografía tridimensional, con los datos capturados con la computadora, y a su vez con estos datos programar un manipulador y remover un área particular de un hueso. Los robots semiactivos y los robots pasivos transforman los movimientos desde un operador en movimientos reforzados por los actuadores del brazo del robot.

En la década de los 70, con el avance de la computación, se logran importantes avances en la industria robótica, principalmente en el área del empaquetado y la soldadura. Hoy los robots están incorporados en todas las áreas productivas de las empresas, existiendo para el año 2012, una venta de robots anual equivalente a 159.346 unidades (18) [28].

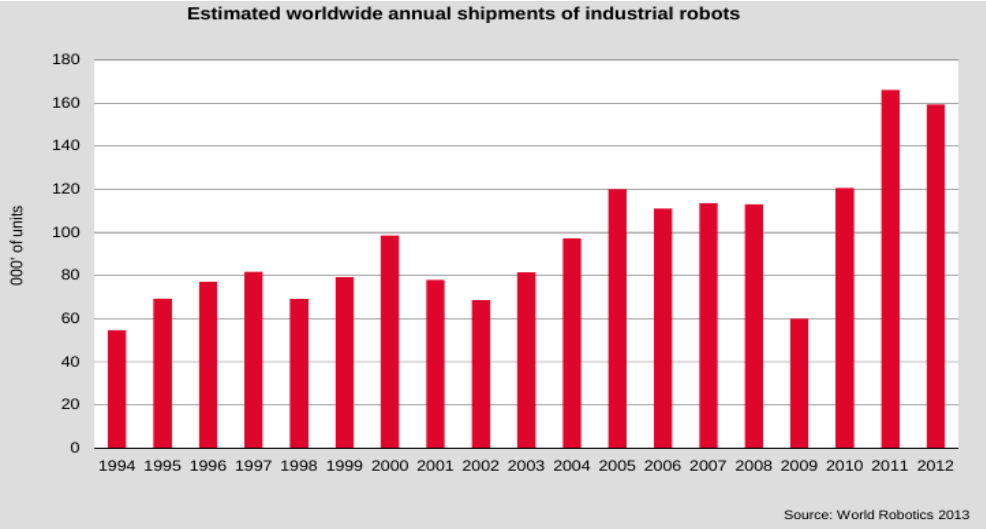



*Ilustración 18: Estadísticas de ventas anuales de robots hasta 2012, fuente [28].*

El parque robótico se encuentra mayormente concentrado en la industria automotriz (ver ilustración 19) donde se requiere de una alta precisión y continuidad, además, al trabajar con materiales pesados, los operadores humanos corren riesgos que pueden ser evitados con el uso de la robótica, en particular de los manipuladores o brazos robóticos o automáticos.

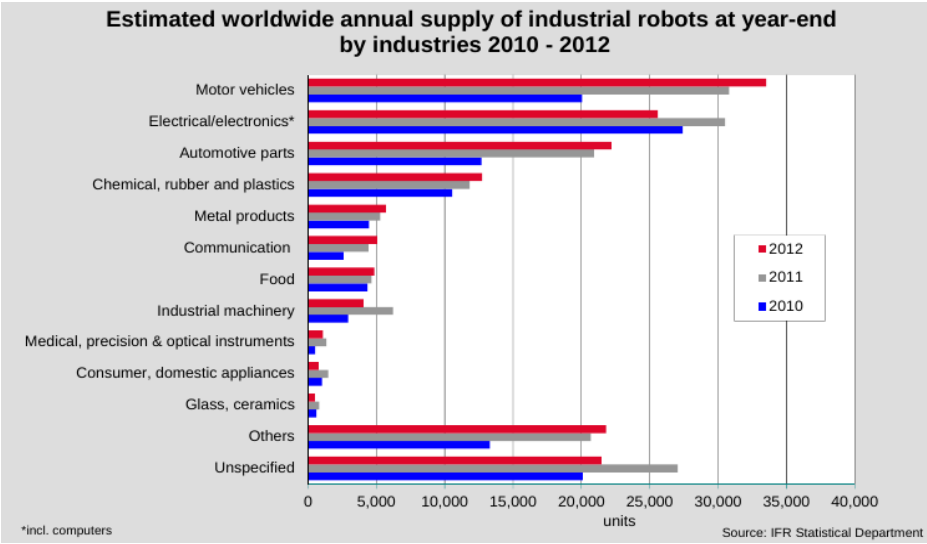



*Ilustración 19: Estadísticas de uso de robots por rubro industrial, fuente [28]*

## Tipos de Robots

De forma general, los robots podrían ser entendidos como robots industriales y robots de servicio, dentro de cada uno de los tipos, existen los robots estáticos y los robots móviles y dentro de cada tipo se pueden entender los autónomos y los teleoperados. No se profundizará mas en la clasificación de

robots puesto que los robots teleoperados son el principal requerimiento en este proyecto.

### *Robótica Industrial:*

Según la Asociación de Industrias Robóticas: Un robot industrial es un manipulador multifuncional reprogramable capaz de mover materias, piezas, herramientas o dispositivos especiales según trayectorias variables, programadas para realizar tareas diversas [29].

Dentro de la robótica industrial, encontramos robots de tipo manipuladores, de repetición o aprendizaje y aquellos controlados por computador. Los **Manipuladores**, son sistemas mecánicos de pocos movimientos y sencillos de controlar, ya sea manualmente con una secuencia de acciones pre-programadas o fijas o a través de secuencias reprogramadas, en cuyo caso se pueden adaptar las funciones del robot en función de sus requerimientos, este tipo de robots es sin duda de los mas difundidos en el mundo industrial por sus capacidades al permitir trabajo repetitivo, y que requiere un elevado nivel de fuerza. Estos manipuladores pueden ser programados y luego instalados o bien controlados de forma remota por un computador, el cual también puede ser capaz de modificar sus funciones y monitorizar su estado. Además pueden ser entrenados utilizando una función de imitación que les permite imitar el comportamiento de un experto, este tipo de robots son también conocidos como robots pintores o de soldadura. En la ilustración 20 se muestra una imagen de un manipulador adquirido por nuestra Universidad.

*Ilustración 20: Manipulador FANUC adquirido por la Universidad*

### *Robótica de Servicio:*

Paralelo a la robótica industrial, existe otra rama de la robótica que no está ligada al sector productivo, es la llamada robótica de servicio, que tiene como objetivo servir al ser humano en tareas cotidianas como la limpieza o en otro tipo de tareas que no tienen que ver con la manufactura, tales como la medicina, la exploración y la entretención. La robótica de servicio puede ser definida como: Un robot

que opera de forma automática o semiautomática para realizar servicios útiles al bienestar del ser humano o a su equipamiento, excluyendo las tareas de fabricación.

Las áreas en las que se están desarrollando los robots de servicio son muy variadas, esto porque ya existe la idea que en un futuro próximo, los robots serán capaces de realizar todas las tareas de tipo físico especialmente las más pesadas o peligrosas. Los sectores económicos que están incorporando la robótica de servicios son: la agricultura, la construcción, la minería, la energía, la seguridad y defensa, la medicina y la vida doméstica a través de la domótica.

Las principales diferencias entre un robot de servicio respecto de un robot industrial son las siguientes:

- Los robots industriales generalmente son brazos robóticos estáticos diseñados para trabajar con alto tonelaje, los robots de servicio por otra parte son bastante mas complejos pudiendo incluso desplazarse por un espacio tridimensional de trabajo.
- Son autónomos, y pueden desenvolverse relativamente bien (esta tecnología aun está en desarrollo) en lugares desconocidos.
- Los robots de servicio son capaces de realizar procesamiento no solo local, poseen una gran variedad de sensores que le permiten percibir el medio ambiente y tomar decisiones en función de sus percepciones.
- La complejidad y falta de estructuración de las tareas que debe realizar le exige a los robots de servicio tener una gran gama de sensores y una gran maniobrabilidad.

### *Telerobótica*

Se han reconocido ampliamente los beneficios que presenta el concepto de operación robótica remota en diversos ámbitos; desactivando bombas, explorando el espacio y el océano profundo y tratando pacientes en el campo de batalla en un terreno seguro detrás de la linea de enfrentamiento, estas son algunas de las aplicaciones potenciales de la telerobótica y que se han puesto en práctica este último tiempo.

La Telepresencia o la inserción de un operador de un robot en un display de realidad virtual, surgen como potenciales beneficios, así como también operaciones virtuales o teleoperación de camiones de carga [30][31][32]. Ya desde los años 80, la National Aeronautics and Space Administration (NASA) se unió con la Ames Research Center para comenzar del desarrollo de un casco de realidad virtual que le permitiera a los usuarios sumergirse en un gran conjunto de datos que eran transmitidos desde las misiones espaciales. Al añadir la visión estereoscópica 3D con DataGlove los usuarios podían ver sus propias interacciones con el mundo virtual.

La telerobótica hoy en día es una de las ramas de la robótica con mayor crecimiento y de mayor interés para los científicos puesto que acercan al ser humano a lugares donde no es posible llegar como el océano profundo o el espacio exterior, existen ya numerosas misiones espaciales que basan su

funcionamiento en la telerobótica.

Los camiones autónomos de la mina Gaby, en Chile, son otro gran ejemplo del desarrollo de la telerobótica llegando al punto de automatizar camiones de enormes tonelaje y con un sistema tan robusto que da la confianza a los ingenieros de la mina para poner un camión autónomo que podría ocasionar un gran desastre si falla.

Pero para satisfacer los requerimientos de la telerobótica se deben cumplir varios requerimientos con el fin de asegurar la calidad, operabilidad y precisión que se requiere a la hora de trabajar con robots que están geográficamente lejos de sus operarios. Primero que todo se debe asegurar una conexión que sea lo suficientemente estable como para soportar un alto trafico de datos, recibir la información de todos los sensores que posee el robot y además enviar información de control para que este tome decisiones cuando sea necesario.

Además de la infraestructura, se debe contar con un software lo suficientemente robusto como para otorgar el mayor tiempo de operabilidad posible, ser lo mas tolerable a fallos y por sobre todo que sea capaz de dejar en un estado seguro al robot luego de alguna falla, o ser capaz de recuperarse automáticamente, cabe recordar que si un robot que esta en otro planeta falla o pierde la conectividad, es prácticamente imposible recuperarlo por lo que son miles de dólares los que se estarían perdiendo.

Una de las tendencias mas fuertes en el último tiempo corresponde al desarrollo de telerobótica a través de Internet [33]–[42], lo cual implica que la transferencia de datos de operación así como también los datos de sensores e información de visualización, es enviada al operador o robot remoto, usando como medio de comunicación la Internet.

El creciente interés por este campo es estimulado además por el creciente avance que existe en la Internet, que provee acceso a muchos recursos computacionales de forma virtual a todos en cualquier lugar del mundo, lo cual significa que un robot podría utilizar un cluster (término informático para conjunto de servidores de alto computo) para tomar decisiones y liberar de poder de cálculo al robot. Otra de las ventajas de utilizar internet como medio de comunicación para telerobótica es el aumento de aplicaciones disponibles así como las constantes mejoras en los medios de comunicación y por sobre todo, el uso de protocolos estándar utilizados en internet, los cuales pueden ser luego aplicados en telerobótica eliminando así la necesidad de desarrollar y mantener protocolos propietarios o especializados. La telerobótica sobre Internet difiere de la telerobótica convencional en que cuando se desee, el robot puede estar disponible para cualquier persona del mundo, usando solo Internet, lo cual permite a personas en otras partes del mundo colaborar y participar activamente en la exploración remota en varias aplicaciones. Las áreas en las que la telerobótica pueden ser utilizadas incluyen: entretenimiento, manufactura remota, medicina remota, operación remota y minería. Además tiene un gran impacto en la investigación donde los laboratorios pueden compartir acceso a recurso costosos. Sin embargo existen problemas como el “limitado” ancho de banda de Internet, particularmente a la hora de enviar datos de visualización, los cuales serán cada vez más grandes en función de la calidad con que se transmita dicha información.

Sin duda alguna, la telerobótica además puede ser utilizada para la teleoperación de observatorios que se encuentran en lugares remotos, considerando para ello, las recomendaciones que hace el mundo de la robótica respecto de la telerobótica y teleoperación.

## Manipuladores Mecánicos

Los manipuladores mecánicos corresponden a una de las mas importantes forma de robótica industrial. Exactamente qué constituye un robot industrial, es un punto a debatir, según [43] los brazos robóticos como el adquirido por la Universidad siempre son incluidos como robots industriales, pero los robots controlados numéricamente generalmente no son incluidos como robots industriales. La diferencia radica en la sofisticación y programabilidad de dichas máquinas, si ésta puede ser programada para realizar una amplia variedad de tareas, entonces puede ser catalogado como robot industrial, sin embargo aquellas máquinas que están diseñadas para realizar un solo tipo de tarea puntual, son llamadas Autómatas **fijos**.

En general, el estudio de la mecánica y el control de manipuladores no corresponde a una ciencia nueva si no mas bien a una colección de temas tomados de las “áreas clásicas”. La Ingeniería Mecánica contribuye con la metodología para el estudio de máquinas estáticas y dinámicas. La Matemática otorga herramientas para describir movimientos espaciales y otros atributos de los manipuladores. La teoría de control entrega herramientas para el diseño y evaluación de algoritmos que permitan realizar los movimientos deseados o aplicar una fuerza determinada. Las técnicas de Ingeniería Eléctrica son aplicadas en la construcción de sensores e interfaces para robots industriales y las ciencias de la computación contribuyen desarrollando la base para poder programar estos dispositivos y realizar una tarea determinada.

Existen diferentes tipos de manipuladores y pueden ser descritos en función de la forma en que se interpretan o se le indican los movimientos (ver ilustración 21).

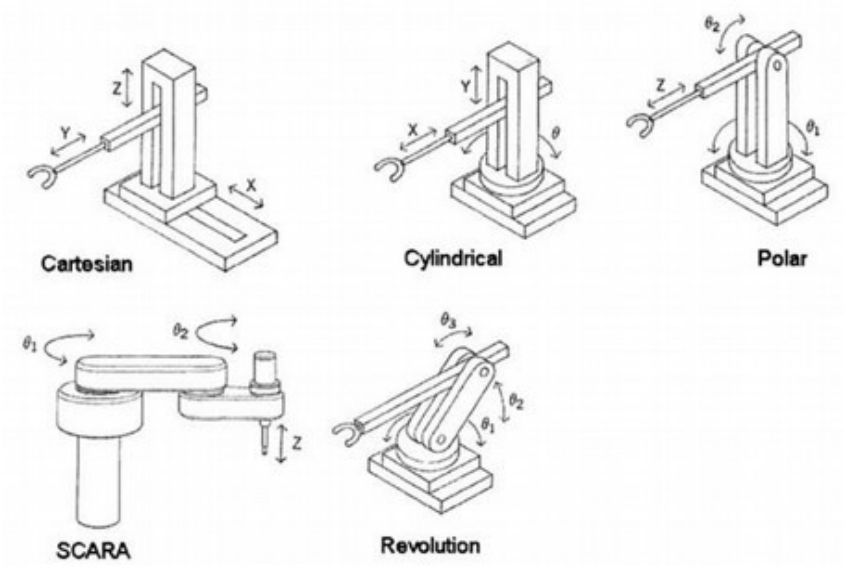



*Ilustración 21: Diferentes tipos de manipuladores según sus coordenadas, fuente: [43]*

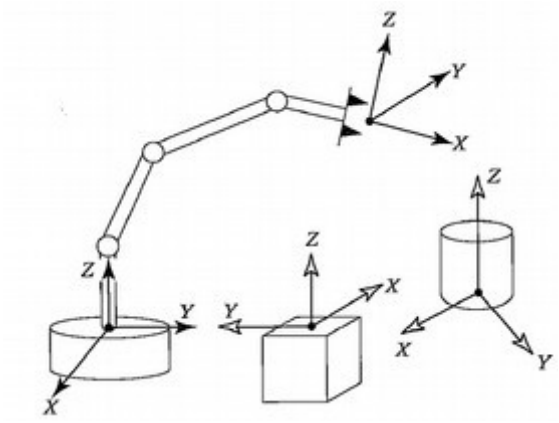
En el estudio de la robótica, es una preocupación, la ubicación de objetos en el espacio tridimensional. Dichos objetos pueden ser las uniones del manipulador, las partes o herramientas con las cuales tiene que lidiar y cualquier otro objeto del ambiente el manipulador. A un nivel rudimentario pero importante, dichos objetos están descritos por solo dos atributos: posición y orientación. Naturalmente, un tópico de interés inmediato es la forma en la cual se representan estas cantidades y como son manipuladas matemáticamente.

Con el fin de describir la posición y orientación de un cuerpo en el espacio, siempre se utilizará un sistema de coordenadas o **frame** al objeto. Luego se procede a describir la posición y la orientación de este **frame** en función de algún sistema de coordenadas.

Cualquier frame puede servir como sistema de referencia en el cual se pueda expresar la posición y orientación del cuerpo, luego se transforma o cambia la descripción de esos atributos de un cuerpo a otro.[43]

### *Cinemática directa de un manipulador*

La cinemática es la ciencia del movimiento, la cual trata el movimiento sin tomar en cuenta las fuerzas que lo causan. Dentro de la ciencia de la cinemática, uno puede estudiar la posición, la velocidad, la aceleración y todas las siguientes derivadas de la variable posición (con respecto al tiempo o a otra variable o variables). Por lo tanto, el estudio de la cinemática de un manipulador hace referencia a todas las propiedades geométricas y basadas en el tiempo del movimiento.



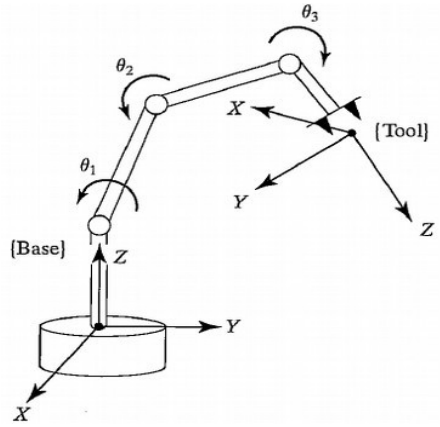


*Ilustración 23: Grados de libertad de un manipulador, fuente: [24]*

El número de **grados de libertad** (ver ilustración 23) que un manipulador posee es el número de posiciones variables independientes que tendrían que ser especificadas para describir y localizar todas las partes del mecanismo. Esto es en general un termino usado para todos los mecanismos. Por ejemplo, una barra con 4 uniones tiene solo un grado de libertad (a pesar de que hay 3 miembros en movimiento). En el caso de un robot industrial típico, debido a que los manipuladores tienen usualmente una cadena cinemática abierta y además porque la posición de cada unión es definida con una sola variable, el número de uniones es igual a los grados de libertad.

En el extremo libre de la cadena de uniones que conforman al manipulador se encuentra el actuador distal (end-effector). Dependiendo del propósito del robot, el actuador distal podría ser un gripper (pinza), un soplete para soldadura, un electroimán o cualquier otro dispositivo compatible o adaptado para el manipulador. Generalmente se describe la posición del manipulador en función de la descripción del frame de la herramienta, el cuál es agregado al final del actuador, y es relativo al frame base, el cual está sujeto a la base no-móvil del manipulador.

Un problema básico en el estudio de la manipulación mecánica es llamada, cinemática directa. Este es un problema de geometría estática y consiste en calcular la posición y la orientación del actuador distal del manipulador. Específicamente, dado un conjunto de ángulos de articulación, el problema de la cinemática directa es calcular la posición y la orientación de la herramienta relativa al frame base.[43]

### *Cinemática inversa de un manipulador*

El problema de la cinemática inversa consiste en lo siguiente: Dadas la posición y orientación del actuador distal del manipulador, calcular todos los posibles conjuntos de ángulos de articulación que podrían ser utilizados para alcanzar la posición y orientación dadas. Este es el problema fundamental en el uso práctico de un manipulador.

Este es un problema geométrico bastante complejo que es resuelto rutinaria y diariamente miles de veces por un humano, y el objetivo en de la cinemática inversa es lograr crear un algoritmo en el

computador de control que sea capaz de realizar estos cálculos. De cierta forma, la solución de este problema corresponde al elemento mas importante en el sistema de un manipulador.

Uno podría pensar que este problema corresponde a un “mapping” de “ubicaciones” en el espacio cartesiano 3D para ubicaciones de las articulaciones internas del robot.. Esta necesidad surge naturalmente cada vez que un objetivo es especificado en coordenadas del espacio 3D exterior. Algunos de los primeros robots carecían de este algoritmo, estos robots eran simplemente movidos (muchas veces con las manos) a las posiciones deseadas, las cuales luego eran guardadas en función de ángulos de articulación (como por ejemplo la ubicación en el espacio de la articulación) para luego ser reproducidas. Obviamente, si el robot es usado únicamente en el modo de grabado y reproducción de las posiciones de las articulaciones y los movimientos, no es necesario un algoritmo relacionado con el espacio de la articulación y el espacio cartesiano. Estos días, sin embargo es raro encontrar un robot industrial que no posea la cinemática inversa básica.

El problema de la cinemática inversa no es tan simple como la cinemática directa. Principalmente porque las ecuaciones son no-lineales, sus soluciones no siempre son fáciles (o siquiera posibles) en una expresión compacta. Además, surgen preguntas como la existencia de alguna solución o de múltiples soluciones. La existencia o no existencia de una solución cinemática define el espacio de trabajo de un determinado manipulador. La falta de soluciones significa que el manipulador no puede alcanzar la posición deseada y la orientación porque esta se encuentra fuera del espacio de trabajo del manipulador.

Además de lidiar con el problema de posicionamiento estático, uno podría querer analizar el movimiento del manipulador. A menudo, cuando se realiza un análisis de velocidad del mecanismo, es conveniente definir una matriz de cantidades llamada **Jacobiano** del manipulador. El jacobiano especifica el mapping desde las velocidades en el espacio de las articulaciones a velocidades en el espacio Cartesiano. La naturaleza de este mapping varía según varie la configuración del manipulador. En ciertos puntos, llamados **singularidades**, este mapping no es invertible. Una singularidad corresponde a un punto en el cual no puede seguir moviéndose en dicha dirección y debe reconfigurar sus otras articulaciones para seguir con el movimiento.

Los manipuladores no siempre se mueven por el espacio, algunas veces además requieren tocar una pieza de trabajo o una superficie de trabajo y aplicar una determinada fuerza. En este caso surge el siguiente problema: Dada una fuerza deseada de contacto y un determinado momento, qué conjunto de torques de articulación es necesario para generarlos?. Una vez mas, la matriz Jacobiana del manipulador surge prácticamente de forma natural en la solución de este problema. [43]

### *Dinámica de un manipulador:*

La dinámica es un tema ampliamente estudiado dedicado a estudiar las fuerzas requeridas para provocar un movimiento. Para acelerar un manipulador desde el reposo, mover el actuador distal a una velocidad constante y finalmente desacelerar para detenerse, un complejo conjunto de funciones de

torque deben ser aplicados por los actuadores de las articulaciones. La forma exacta de las funciones requeridas por el par de torsión del actuador dependen de los atributos espaciales y temporales de la ruta tomada por el actuador distal a de las propiedades de masa de las uniones y de la carga útil, fricción en las articulaciones, etc. Un método para controlar el manipulador y que siga una ruta deseada, consiste en calcular el torque de los actuadores usando las ecuaciones dinámicas de movimiento del manipulador.

Muchos de nosotros hemos tenido la experiencia de tomar un objeto. que es de hecho mucho mas liviano de lo esperado. Dichos juicios erróneos acerca de la carga pueden provocar un movimiento de levantamiento inusual. Este tipo de observación indica que el sistema de control humano es mas sofisticado que tan solo un esquema puramente cinemático. Mas bien, nuestro sistema de control de manipulación hace uso del conocimiento de la masa y otros efectos dinámicos. Igualmente, los algoritmos que se construyen para controlar el movimiento de un robot manipulador, debieran tomar en cuenta la dinámica.

Un segundo uso de las ecuaciones de dinámica de movimiento es en **simulación**. Reformulando las ecuaciones de dinámica, la aceleración puede ser calculada en función del torque de los actuadores, por lo tanto es posible simular como un manipulador podría moverse en una situación particular en función de un conjunto de torques de los actuadores. Como el poder de computo es mucho mas rentable, el uso de simulaciones está creciendo en uso e importancia en varios ámbitos. [43]

### *Generación de trayectoria.*

Una forma común de hacer que un manipulador se mueva desde un lugar a otro de una manera suave es provocar en cada articulación un movimiento suave usando una función suave a través del tiempo. Comúnmente cada articulación comienza y termina su movimiento al mismo tiempo, con lo cual el movimiento del manipulador parece coordinado. Exactamente como calcular esos movimientos es el problema de la **generación de trayectoria**.

Frecuentemente, una ruta es descrita no solo por el destino deseado sino que también por algunas ubicaciones intermedias o **puntos de vía** (via points) por los cuales el manipulador debe pasar para llegar al destino. En dichas instancias el término **spline** es algunas veces usado para referirse a una función suave que pasa a través de un conjunto de puntos de vía.

Para forzar al actuador distal a seguir una linea recta (u otra forma geométrica) por el espacio, el movimiento deseado debe ser convertida a un conjunto equivalente de movimientos de articulaciones. [43]

### *Control de posición Lineal:*

Algunos manipuladores están equipados con motores steppers (motores paso a paso) u otro actuadores que puedan ejecutar directamente la trayectoria deseada. Sin embargo, la gran mayoría de los manipuladores son impulsados por actuadores que proveen una fuerza o torque para lograr el

movimiento de las uniones. En este caso, un algoritmo es necesario para calcular el torque que provocará el movimiento deseado. El uso de la dinámica es fundamental en el diseño de estos algoritmos, pero por si mismos no constituyen una solución. La primera preocupación de un **sistema de control de posición** es compensar automáticamente los errores en el conocimiento de los parámetros del sistema y eliminar las perturbaciones que tienden a sacar al sistema de la trayectoria deseada. Para lograr esto, los sensores de posición y velocidad son monitorizados por el algoritmo de control, el cual calcula los comandos de torque para los actuadores. [43]

### *Control de posición no Lineal:*

Aunque los sistemas de control basados en modelos de aproximaciones lineales son muy populares en los robots industriales actuales, es importante considerar la dinámica no lineal completa del manipulador cuando se conciben los algoritmos de control. Algunos robots industriales están siendo configurados para el uso de algoritmos de control no lineal en sus controladores. Estas técnicas de control no lineal para los manipuladores prometen mejor rendimiento que los controladores lineales. [43]

### *Control de Fuerza:*

La habilidad de un manipulador para controlar las fuerzas de contacto cuando este toca alguna parte, herramienta o trabaja sobre superficies, son de gran importancia cuando se utilizan los manipuladores en muchas tareas del mundo real. El control de fuerza es complementario al control de posición. Cuando un manipulador se está moviendo por un espacio libre, solo el control de posición tiene sentido porque no hay una superficie de contacto a la cual reaccionar en contra. Cuando un manipulador está tocando una superficie rígida, no obstante, los esquemas de control-posición pueden provocar fuerzas excesivas en el contacto o causar una perdida de contacto con la superficie cuando fue requerida por algún trabajo. Los manipuladores están raramente restringidos por superficies de reacción en todas las direcciones simultaneamanente, por lo tanto una mezcla o control híbrido es requerido, con algunas direcciones controladas por la ley del control de posición y el resto de las direcciones controladas por las leyes del control de fuerza. Un robot debiera ser instruido para lavar una ventana manteniendo cierta fuerza en una dirección perpendicular al plano del vidrio, mientras sigue una trayectoria de movimiento en direcciones tangentes al plano. Este contar con estas especificaciones y con una división o control híbrido son naturales para realizar dichas tareas. [43]

# Framework

En palabras simples, un framework es un conjunto de clases (en el caso de frameworks orientados a objeto, que es el tipo de framework que se utilizará en este trabajo) que incorporan un diseño abstracto para dar solución a una familia de problemas relacionados [44]. Un framework típicamente consiste en una mezcla de clases abstractas y concretas por lo que además son conocidos como librería de clases. Las clases abstractas generalmente residen en el framework mientras que las clases concretas en la aplicación. Entonces, una framework es una aplicación semi-completa, que posee ciertos aspectos fijos comunes a todas las aplicaciones en el domino del problema, junto con ciertos aspectos variables únicos de cada aplicación desarrollada con el framework. En otras palabras, el framework indica un diseño arquitectónico fundamental para las aplicaciones incorporadas en el framework. Además captura la experiencia en programación para resolver una determinada clase de problemas, generalmente los programadores compran o reusan estos frameworks para hacerse de la solución a estos problemas sin tener que resolverlos ellos mismos.

Un framework, predefine muchos parámetros de diseño y acciones, con lo cual el diseñador de la aplicación puede concentrarse en lo específico. Los aspectos variables, llamados "**hot spots**" definen aquellos aspectos de una aplicación, que se deben mantener flexibles para lograr diferentes adaptaciones del framework. Lo que diferencia una aplicación del framework de otra, respecto de un dominio de problema común, es la forma en la que esos "hot spots" han sido especificados.

Sin embargo, los framework aún siguen siendo difíciles de utilizar, el usuario del framework debe entender las complejas clases jerárquicas y las colaboraciones entre objetos diseñadas en el framework para usarlos de la forma correcta y efectiva. Por otro lado, los frameworks son difíciles de documentar debido a que representan una alto nivel de abstracción en las clases del framework y un diseño abstracto [45], [46].

## Dominios de un framework:

Los dominios de problema que un framework puede abarcar son: funciones para una aplicación, funciones para un dominio y funciones de soporte.

### *Frameworks de aplicación:*

Encapsulan la experiencia, aplicable a una gran variedad de programas. Estos frameworks abarcan una linea horizontal de funcionalidad, que pueden ser aplicadas a través de clientes de dominio. Las aplicaciones para GUI (Graphical User Interface) usan frameworks, para dar acceso a todas las funcionalidades de las GUI y lograr el desarrollo de nuevas aplicaciones utilizando dicho framework gráfico para dibujar en la pantalla, con lo cual el desarrollador no debe utilizar tiempo extra en el desarrollo de la interacción entre componentes gráficos y solo dedicarse a desarrollar lo que quiere desplegar gráficamente.

### *Frameworks de dominio:*

Encapsula experiencia en un dominio de problema particular. Estos frameworks, abarcan una linea vertical de funcionalidad que puede ser aplicada a través de clientes de dominio particular. Ejemplos de frameworks de dominio incluyen frameworks de sistemas de control para desarrollar aplicaciones de manufactura, control robótico (como ROS) o control industrial, frameworks para intercambios de información segura, frameworks multimedia para visualización de videos, imágenes, etc. o frameworks para acceso a la red.

### *Frameworks de soporte:*

Proveen una nivel de servicios de sistema, como acceso a archivos, computación distribuida o drivers para dispositivos. Los desarrolladores de aplicaciones utilizan los frameworks directamente o bien utilizan las modificaciones hechas por los proveedores de sistemas. Sin embargo, incluso estos frameworks pueden ser personalizados.

## Estructura de un Framework

Identificar la estructura de alto nivel del framework, puede hacer mas fácil la descripción del comportamiento del framework y provee un punto de inicio para el diseño de interacciones con el framework.

Por ejemplo, algunos frameworks pueden ser descritos como manager-driven, lo que significa que una sola función ejecuta la mayoría de las acciones del framework. Se hace un llamado a la función controladora, para iniciar el framework y éste se encarga de crear los objetos necesarios y ejecutar las funciones necesarias para cumplir con una tarea específica. Un framework de aplicación, generalmente usa un objeto administrador para tomar los eventos de entrada desde el usuario y luego distribuirlo dentro del framework.

Otra caracterización es en función de cómo el framework es utilizado, es decir si derivan clases nuevas o se instancian y se combinan las clases existentes. Esta distinción, es algunas veces referida como “architecture-driven” versus “data-driven”, o también “inheritance-focused” versus “composition-focused”.

Los frameworks dirigidos por arquitectura dependen de la herencia para realizar la personalización o adaptación. El cliente adapta el comportamiento del framework usando diferentes combinaciones de objetos. El objeto que el cliente pasa al framework, afecta lo que el framework hace, sin embargo el framework define como los objetos pueden ser combinados.

Los frameworks que son altamente dirigidos por la arquitectura pueden ser difíciles de usar porque requieren que el cliente desarrolle una gran cantidad de código antes que se puedan producir resultados y comportamientos interesantes. Los frameworks dirigidos por los datos son generalmente fáciles de usar pero pueden ser limitantes.

Además los frameworks pueden ser categorizados como verticales y horizontales. Un framework

horizontal provee servicios a nivel de sistema como acceso a archivos o drivers de dispositivos. Son usualmente independientes del dominio y proveen soporte para muchos otros frameworks verticales que se encuentran en la parte superior del framework horizontal. Un ejemplo de framework horizontal son las API de desarrollo para android (puesto que otorgan una completa capa de software donde son construidas el resto de aplicaciones), mientras que un framework vertical podría ser un framework de control puesto que los framework verticales son conocidos como de aplicación.

## Computación Distribuida

La computación distribuida corresponde a una rama de la computación encargada de lograr resolver problemas utilizando una cantidad masiva de computadores, de esta forma, lo que se busca es unir el poder de computo de varios computadores para lograr tener un solo gran computador funcionando. Estos computadores se comunican entre si a través de algún protocolo de comunicación (generalmente TCP/IP) a altas velocidades.

En particular, se verá el concepto de Computación Distribuida como un conjunto de computadores trabajando colaborativamente entre sí. Es decir, cada computador de la red de computadores que constituyen el sistema distribuido, cumple una labor en particular, pero para ojos de un usuario todos estos sistemas representan uno.

La computación distribuida tienen un requisito fundamental, una arquitectura de telecomunicaciones muy robusta, y una alta tolerancia a fallos, con esto se asegura la operabilidad en todo momento de la plataforma que depende de este sistema distribuido.

## CORBA

Common Object Request Broker Architecture (CORBA) es un estándar definido por Object Management Group (OMG)[46], [47] que permite que diversos componentes de software escritos en múltiples lenguajes de programación y que corren en diferentes computadoras, puedan trabajar juntos; es decir, facilita el desarrollo de aplicaciones distribuidas en entornos heterogéneos.

CORBA fue el primer producto propuesto por OMG. Su objetivo es ayudar a reducir la complejidad, disminuir los costes y acelerar la introducción de nuevas aplicaciones informáticas, promoviendo la teoría y la práctica de la tecnología de objetos en los sistemas distribuidos.

Es una tecnología que oculta la programación a bajo nivel de aplicaciones distribuidas. No obstante también brinda al programador una tecnología orientada a objetos; las funciones y los datos se agrupan en objetos y estos objetos pueden estar en diferentes máquinas, pero el programador accederá a ellos a través de funciones normales dentro de su programa.

CORBA es más que una especificación multiplataforma, también define servicios habitualmente necesarios como seguridad y transacciones y así éste no es un sistema operativo en sí, en realidad es un middleware.

# ROS

ROS (Robot Operating System) [21], [48], [49] es un proyecto de código abierto y un meta-sistema operativo para robots. ROS Provee los servicios que se esperarían de un sistema operativo incluyendo abstracción de hardware, control de dispositivos de bajo nivel, implementación de funcionalidades comúnmente utilizadas, traspaso de mensajes entre procesos y administración de paquetes. Además provee herramientas y librerías para obtener, construir, escribir o ejecutar código entre múltiples computadores. ROS es similar en ciertos aspectos a "frameworks robóticos" como Player [50], YARP [51], Orocos [52], CARMEN [53], Orca [54], MOOS [55] y Microsoft Robotics Studio [56].

El grafo de ejecución de ROS corresponde a una red peer-to-peer de procesos que están acoplados flexiblemente, usando la infraestructura de comunicación de ROS. Este framework implementa varios diferentes estilos de comunicación, incluyendo llamados a procedimientos remotos de forma síncrona usando servicios, transmisión asíncrona de datos a través de tópicos y almacenamiento de datos en un servidor de parámetros, estructuras que serán comentadas en la sección Grafo Computacional de ROS

ROS no es un framework de tiempo real, aunque es posible integrar ROS con código de tiempo real. El robot PR2 de Willow Garage, usa un sistema llamado "pr2_etherCAT", que transporta los mensajes de ROS dentro y fuera de un proceso en tiempo real.

ROS no es un framework con la mayor cantidad de características, en vez de eso, el objetivo de ROS es fomentar y soportar el reuso de código en la investigación y desarrollo de la robótica. ROS es un framework de procesos distribuidos que permite a los ejecutables ser diseñados individualmente y acoplados débilmente en tiempo de ejecución. Dichos procesos pueden ser agrupados en Paquetes o "Stacks", los cuales pueden ser fácilmente compartidos o distribuidos. ROS además mantiene un repositorio de código federado que permite la colaboración distribuida.

Para mantener el objetivo de intercambio y colaboración, existen algunas caractersiticas que el framework cumple:

- Ligero: ROS esta diseñado para ser tan liviano como sea posible, por lo tanto el código desarrollado para ROS puede ser utilizado con otro framework de robótica. Un corolario de esto es que ROS es fácil de integrar con otros frameworks y software de robótica. De hecho ROS a sido exitosamente integrado con OpenRAVE, Orocos y Player.
- ROS-agnostic libraries: El modelo de desarrollo deseado es escribir librarias que no dependen de ROS y que puedan ser utilizadas en cualquier parte, con interfaces funcionales claras.
- Independencia del Lenguaje: Desarrollar en ROS es fácil, puesto que se puede trabajar en varios lenguajes de programación modernos, de hecho ROS tiene actualmente implementados Python, C++ y Lisp, además a modo de prueba se tienen de forma experimental librerías en Java y Lua.
- Fácil de probar: ROS tiene un framework de pruebas unitarias y de integración llamado "rostest" que facilita agregar y quitar características de prueba.

- Escalable: ROS es apropiado para grandes sistemas y para grandes grupos de desarrollo.

## Conceptos de ROS

ROS tiene 3 niveles de concepto, Sistema de Archivos, Grafo Computacional y Comunidad. Además de los 3 niveles de concepto mencionados anteriormente, ROS define 2 tipos de nombre: nombres de paquetes y nombres, ambos explicados en la siguiente sección.

### *Nivel de Sistema de Archivos*

El nivel de sistema de archivos principalmente cubre recursos que generalmente se encuentran en disco, tales como:

- **Paquete:** Los paquetes son la unidad principal para organizar el software en ROS. Un paquete pueden contener procesos de ejecución (nodos), una librería dependiente de ROS, un conjunto de datos, archivos de configuración, o cualquier cosa que sea útil organizarla junta. Los paquetes son el ítem mas atómico en ROS. Lo que significa que la unidad mas granular que se puede construir y desarrollar es un paquete.
- **Metapaquete:** Los metapaquetes son paquetes especializados, que solo sirven para representar un grupo de paquetes relacionados.
- Manifiestos de Paquetes: Los manifiestos (package.xml), proveen meta-información acerca del paquete, incluyendo el nombre, versión, descripción, licencia, dependencias, y cualquier otra meta-información como paquetes exportados.
- **Repositorios:** Una colección de paquetes que comparten un sistema de control de versiones común. Los paquetes que comparten un sistema de control de versiones, comparten la misma versión y pueden ser liberados juntos, usando la herramienta automática “catkin”. Los repositorios pueden contener solo un paquete.
- **Tipos de mensajes (msg):** Descripciones de mensajes, guardados en *mi_paquete/msg/MiTipoDeMensaje.msg*, definen la estructura de datos para los mensajes enviados en ROS. Los mensajes corresponden a datos enviados a través de la infraestructura de comunicación de ROS.
- **Tipos de servicios (srv):** Descripciones de servicios, guardadas en *mi_paquete/srv/MiTipoDeServicio.srv* definen la estructura de datos de las peticiones y las respuestas para los servicios en ROS. Debido a que en ROS se utiliza la arquitectura de publicador/subscriptor, un servicio es la instancia para publicar datos para que un subscriptor pueda obtenerlos.

### *Nivel de Grafo computacional*

El Grafo Computacional, es un red peer-to-peer de procesos de ROS que están procesando datos juntos.

Los conceptos básicos del Grafo Computacional de ROS son nodos, Maestro, Parámetro, Servidor, Mensajes, Servicios, Tópicos y Bolsas, todo ellos proveen datos al Grafo de diferentes formas.

Todos estos conceptos están implementados en el repositorio ros_comm:

- **Nodo:** Los nodos son procesos que realizan cálculo. ROS está diseñado para ser modular a una escala de grano fino; un sistema de control robótico usualmente entiende a muchos nodos. Por ejemplo, un nodo controla un medidor láser, un nodo controla los motores de las ruedas, un nodo realiza la localización, un nodo ejecuta la planificación de ruta, un nodo provee una representación gráfica del sistema y así. Un nodo de ROS está desarrollado utilizando la librería de cliente de ROS, tales como roscpp (para c++) y rospy (para python).
- **Maestro:** El “ROS Master” provee el nombre de registro y la búsqueda por el resto del Grafo Computacional. Sin el Maestro los nodos no serían capaces de encontrarse entre ellos, intercambiar mensajes e invocar servicios.
- **Servidor de Parámetros:** El servidor de parámetros, permite a los datos ser guardados con una llave en una ubicación central. Éste es actualmente parte del Maestro.
- **Mensajes:** Los nodos se comunican entre si pasándose mensajes entre si. Un mensaje es una estructura de datos simple, que entiende campos con tipos de datos. Las primitivas estándar (integer, floating point, boolean, etc.) son soportadas, así como también listas de tipos primitivos. Los mensajes pueden incluir estructuras de datos anidadas y listas (algo parecido a las estructuras de C).
- **Tópicos:** Los mensajes son enrutados a través de un sistema de transporte que publica/subscribe. Un nodo envía un mensaje publicándolo en un determinado tópico. El tópico es el nombre usado para identificar el contenido del mensaje. Un nodo que esta interesado en cierto tipo de dato, se va a subscribir al tópico que necesite revisar. Pueden haber múltiples publicadores y subscriptores concurrentemente para un solo tópico y un solo nodo puede publicar y/o suscribirse a múltiples tópicos. En general, los publicadores y suscriptores no se preocupan de la existencia de cada uno. La idea es desacoplar la producción de información para su consumo. Lógicamente, uno podría pensar en un tópico, como un bus de mensajes fuertemente tipado. Cada bus tiene un nombre, y cualquiera puede conectarse al bus y enviar o recibir mensajes siempre y cuando sea del tipo de dato correcto.
- **Servicios:** El modelo de publicador/suscriptor, es un paradigma de comunicación muy flexible, pero es mucho-a-mucho, un transporte de mensajes solo de ida no es apropiado para interacciones del tipo petición/respuesta, los cuales son frecuentemente requeridos en los sistemas distribuidos. Las peticiones/respuestas son realizadas a través de los servicios, los cuales son definidos por un par de mensajes: un tipo de mensaje para la petición y uno para la respuesta. Un nodo productor ofrece un servicio bajo un nombre y el cliente usa el servicio enviando un mensaje de petición, esperando por la respuesta. La librería de cliente de ROS

generalmente presenta esta interacción al programador como si se tratase de una llamada a un procedimiento remoto.

- **Bolsas:** Las bolsas son un formato para guardar y replicar datos de mensajes de ROS. Las bolsas son mecanismos importantes para guardar datos como sensores que pueden ser difíciles de recolectar pero que son necesarios para el desarrollo y la prueba de los algoritmos.

El “ROS Master” actúa como un servidor de nombres en el Grafo Computacional de ROS. Este guarda la información de registro de tópicos y servicios para nodos de ROS. Los nodos se comunican con el Maestro para reportar su información de registro. Como estos nodos se comunican con el Maestro, estos pueden recibir información acerca de otros nodos registrados y realizar apropiadamente las conexiones. El maestro incluso hará llamadas internas a dichos nodos, cuando su información de registro cambie, lo que permite a los nodos, crear dinámicamente conexiones cuando nodos nuevos se ejecuten.

Los nodos se conectan a otros nodos directamente; el maestro solo provee información de búsqueda, algo parecido a un servidor DNS. Los nodos que se suscriben a un nodo requieren conexiones de nodos que publiquen en ese tópico, y se establecerá una conexión sobre un protocolo acordado. El protocolo mas común en ROS es llamado TCPROS, el cual usa sockets TCP/IP estándar.

Esta arquitectura permite el funcionamiento desacoplado, donde los nombres (que pueden ser jerárquicos) son el componente principal para la construcción de sistemas mas grandes y complejos. Los nombres tienen un rol muy importante en ROS puesto que los nodos, tópicos, servicios y parámetros tienen nombre. Cada cliente ROS soporta renombrar vía linea de comando, lo que significa que un programa compilado puede ser reconfigurado en tiempo de ejecución para utilizar una topología computacional diferente. En la 24 se puede apreciar un diagrama que explica como funciona la arquitectura de ROS.

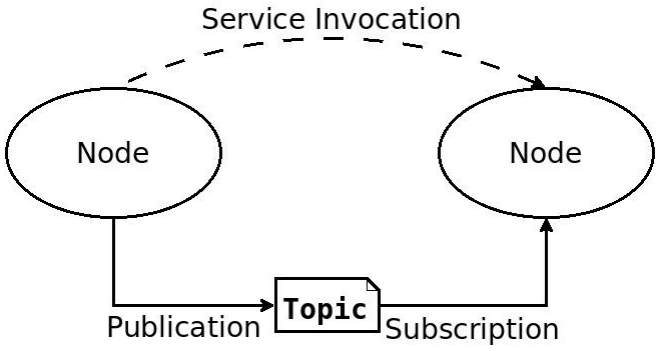



*Ilustración 24: Funcionamiento básico de ROS*

### *Nivel de Comunidad de ROS*

El concepto de nivel de Comunidad de ROS consiste en recursos de ROS que permite a comunidades separadas intercambiar software y conocimiento. Estos recursos incluyen:

- **Distribuciones:** Una Distribución de ROS es un conjunto de stacks que se pueden instalar. Las Distribuciones juegan un papel similar a las distribuciones Linux hacen mas fácil la instalación de una colección de software y se mantiene una versión consistente entre el conjunto de software.
- **Repositorios:** ROS depende de una red federada de repositorios de código, donde diferentes instituciones pueden desarrollar y liberar sus propios componentes de software para robots.
- **La Wiki de ROS:** La Comunidad de la Wiki de Ros es un foro principal para documentar información acerca de ROS. Cualquiera puede crearse una cuenta y contribuir con documentación, realizar correcciones o actualizaciones, escribir tutoriales y mas.
- **Listas de Correo:** Corresponde al canal de comunicación primario para nuevas actualizaciones de ROS, así como también puede ser utilizado como un foro para preguntar acera de ROS.
- **ROS Answer:** Sitio web de preguntas y respuestas

### *Nombres*

#### Grafo de Nombre de Recursos

El “Graph Resource Name” o Grafo de Nombre de Recursos, provee una estructura de nombre jerárquica que es usada para todos los recursos en el Grafo Computacional de ROS, estos recursos corresponden a los Nodos, Parámetros, Tópicos y Servicios. Estos nombres son una forma muy poderosa en ROS y es muy importante para entender cómo grandes y complicados sistemas basados en ROS están compuestos, por lo tanto es crítico entender cómo funcionan los nombres y cómo pueden ser manipulados

Antes de describir mas nombres, a continuación se muestran algunos ejemplos:

- */ (el namespace global)*
- */foo*
- */ucn/robot/madeusa*
- */disc/boebot*

El Grafo de Nombre de Recursos, es un mecanismo importante en ROS para entregar encapsulación. Cada recurso es definido dentro del namespace, el cual puede ser compartido con muchos otros recursos. En general, los recursos pueden crear recursos dentro de su namespace y pueden acceder a estos recursos dentro o sobre su propio namespace. Las conexiones pueden ser hechas entre recursos en

diferentes namespace, pero esto es hecho generalmente por código de integración encima de ambos namespace. Esta encapsulación aísla diferentes porciones del sistema, de usar el recurso equivocado o utilizar nombres globales.

Los nombres son resueltos de forma relativa, por lo tanto los recursos no deben preocuparse en que namespace se encuentran. Esto simplifica la programación de nodos, entendiéndolos como programas que trabajan juntos tal y como si estuvieran en el nivel mas alto del namespace. Cuando estos nodos son integrados a sistemas mas grandes, estos pueden ser empujados hacia abajo dentro de un namespace que define esta colección de código. En palabras simples con cada namespace, se genera una especie de paquete digital que aísla porciones de código, luego se puede generar un namespace superior que encapsule estos dos paquetes digitales, con lo cual eventualmente se podrían tener dos namespaces que poseen los mismos nombres de paquetes pero con funcionalidades diferentes y no causarían conflictos. Por ejemplo, se podría tener el namespace *boebot* y el namespace *lego*, ambos namespace podrían contener un paquete llamado *movement* y otro llamado *artificial_intelligence*, ambos paquetes podrían luego ser encapsulados bajo el namespace *robot* y luego es evidente que a pesar de tener nombres iguales, la forma de llegar a ellos a través del Grafo es diferente, por lo tanto constituyen softwares diferentes.

### Nombres Válidos

Un nombre válido tiene las siguientes características:

1. El primer carácter es carácter alfabético ([a-z|A-Z]), tilde (~) o slash (/)
2. Los caracteres subsiguientes pueden ser alfanuméricos ([0-9|a-z|A-Z]), guión bajo (_), o slash (/)

*Excepción: Los nombres bases (descritos mas abajo) no pueden tener slash (/) o tildes (~) en ellos.*

### Resolviendo Nombres

Hay 4 tipos de Nombres en ROS: base, relativo, global y privado, los cuales tienen la siguiente sintaxis respectivamente:

- *base*
- *nombre/relativo*
- */nombre/global*
- *~nombre/privado*

Por defecto, la resolución de nombres es hecha de forma relativa al namespace del nodo. Por ejemplo el nodo */disc/nodo1*, esta bajo el namespace */disc*, por lo tanto el *nodo2* será resuelto como */disc/nodo2*

Los nombres sin namespace de ningún tipo, son nombres base. Los nombres base son de hecho una subclase de nombres relativos y tienen las mismas reglas de resolución. Los nombres base son

frecuentemente usados para inicializar el nombre del nodo.

Nombres que comienzan con "/" son globales – estos son considerados como completamente resueltos. Los nombres Globales deberían ser evitados tanto como sea posible porque limitan la portabilidad del código.

Nombres que comienzan con "~" son privados. Estos convierten el nombre del nodo dentro del namespace. Por ejemplo, *nodo1* en el namespace */disc* tiene el namespace privado */disc/nodo1*. Los nombres privados son útiles para pasar parámetros a nodos específicos a través de un servidor de parámetros.

A continuación se muestran algunos ejemplos de resolución de nombres.

| Nodo | Relativo | Global | Privado |
|---|---|---|---|
| /nodo1 | bar → /bar | /bar → /bar | ~bar → /nodo1/bar |
| /disc/nodo2 | bar → /disc/bar | /bar → /bar | ~bar → /disc/nodo2/bar |
| /disc/nodo3 | foo/bar → /disc/foo/bar | /foo/bar → /foo/bar | ~foo/bar → /disc/nodo3/foo/bar |

### *Nombres de recurso de Paquetes*

Los nombres de recurso de paquetes en ROS son utilizado en ROS con el nivel de sistema de archivos para simplificar el proceso de referenciar archivos y tipos de datos en el disco. Los nombres de recurso de Paquetes son son muy simples: estos son solo el nombre del paquete en el que se encuentra el recurso, mas el nombre del recurso. Por ejemplo, el nombre "std_msgs/String" hace referencia al tipo de mensaje "String" en el paquete "std_msgs".

Algunos de los files relacionados con ROS que pueden ser referenciados usando el nombre de recurso de paquete incluyen:

- Los tipos de Mensajes (Message – msg)
- Los tipos de Servicio (Service – srv)
- Los tipos de Nodo (Node)

Los nombres de recurso de paquete, son muy parecidos a las rutas de los archivos, excepto que son mucho mas cortos. Esto se debe a la habilidad de ROS de localizar paquetes en el disco y hacer suposiciones adicionales acerca de su contenido. Por ejemplo, la descripción de los mensajes están siempre almacenadas en el subdirectorio *msg* y siempre tienen una extensión de archivo *.msg* por lo tanto *std_msgs/String*s es una taquigrafía para *ruta/a/std_msgs/msg/String.msg*. De forma parecida, el tipo de nodo *foo/bar,* es equivalente a buscar por un archivo llamado *bar* en el paquete *foo* con permisos de ejecución.

### Nombres Validos

Un nombre de recurso de paquete, tiene estrictas reglas de nombrado que son frecuentemente utilizadas en la auto-generación de código. Por esta razón, un paquete de ROS no puede tener caracteres especiales mas allá del guión bajo y deben necesariamente comenzar con un carácter del alfabeto. Un nombre válido tiene las siguientes características:

- El primer carácter es una letra del alfabeto ([a-z|A-Z)
- Los caracteres subsiguientes pueden ser alfanuméricos ([0-9|a-z|A-Z), guión bajo (_) o un slash (/)
- Tiene que tener al menos un slash (/)

# CAPÍTULO III

## Revisión del estado del Arte

*"Una de las tareas más difíciles de un observatorio en general, es lograr el movimiento fino de un objeto muy grande y pesado y mantenerlo siempre en el foco del CCD [55]"*

---

*En este capítulo se realiza una revisión exhaustiva acerca de software ligado a la Astronomía, se revisan software de tipo profesional como VLTSW y ACS de los observatorios mas grandes del mundo, así como también una gran cantidad de artículos acerca de implementaciones de observatorios Universitarios, el software utilizado y los problemas enfrentados a la hora de implementar y operar un observatorio para finalmente, analizar los software libres/comunes actualmente utilizados en astronomía.*

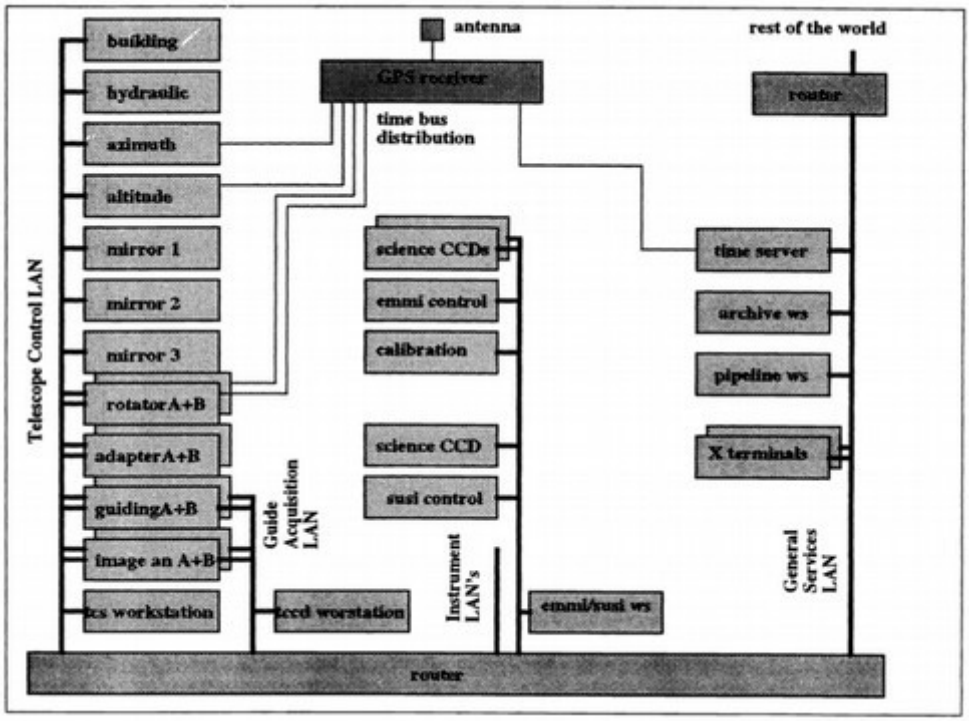
# Estado del Arte

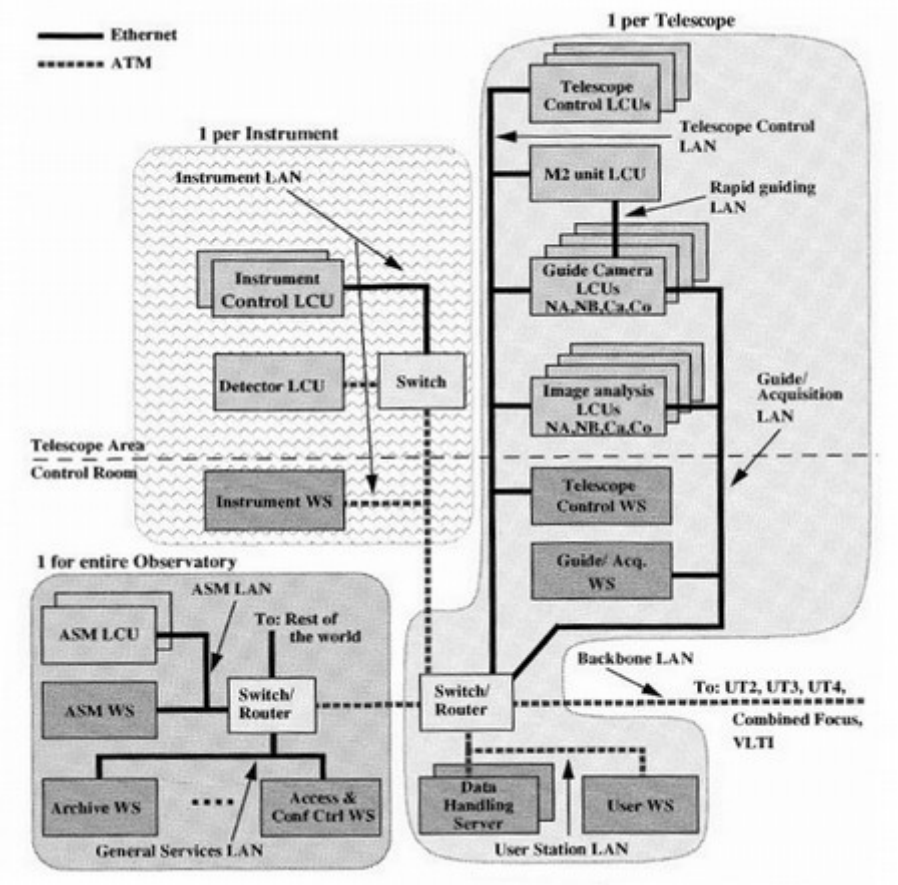
Una de las tareas más difíciles de un observatorio en general, es lograr el movimiento fino de un objeto muy pesado [11], para ello se tienen diversos sistemas de control y diversas arquitecturas utilizadas para el control del telescopio

## Software Profesional de Astronomía

Partiendo el análisis de Sistemas de Control y la arquitectura de software de observatorios, se hará mención a un software de control de la ESO (European Southern Observatory) y del proyecto ALMA (Donde la ESO también participa) principalmente porque la ESO posee y participa en los 3 observatorios mas grandes del mundo; Observatorio Cerro Paranal, E-ELT y ALMA.

El año 1989, la ESO puso en producción el NTT, un observatorio que luego se convertiría en caso de prueba para el Software del Very Large Telescope (VLT) – Very Large Telescope Software (VLTSW). El New Technology Telescope (NTT) [57] es un observatorio con un telescopio de 3.54m de diámetro con montura altazimutal y pionero en óptica activa. En este observatorio, se pueden observar que la estructura con la que se controla el telescopio y los instrumentos, responde a un sistema distribuido basado completamente en Linux y VxWorks principalmente, en el cual se encuentra como Front-end del instrumento, la Instrument WorkStation (IWS), que es un servidor que recibe las peticiones y/o conexiones para luego conectar a las Local Control Unit (LCU) e indicar las acciones que debe realizar en función de las peticiones que recibe, por ejemplo, si la IWS recibe un comando para mover el azimut de un punto **A** a un punto **B**, luego se le envía una o varias ordenes a la LCU para que ésta interactúe con el hardware que está dedicado a controlar (motores, servos, instrumentos, etc.). Todas estas IWS son accesibles a través de X-Terminals, equipos que se conectan a un servidor X (servicio que provee de un display gráfico a las terminales) y desde estas terminales se abre un terminal remoto. Así, son capaces de conectarse gráficamente a las IWS. Esta arquitectura está aún vigente en el observatorio Paranal. Hay diferentes redes locales que funcionan a través de un switch y finalmente un router que conecta las redes cuando esto es necesario. Además, en el NTT hay un conjunto de subredes determinadas para mejorar el rendimiento de la red, debido a un análisis hecho al momento de hacer la implementación. Sin embargo según el mismo artículo, las redes ethernet no proveen la suficiente velocidad, por lo que los sistemas luego serían migrados a tecnologías como FDDI, fiberchannel y ATM. En la ilustración 25 se puede observar la estructura lógica que poseía en ese entonces el NTT. Según consta en [57], ya en ese entonces se pensaba en computación distribuida, pero no a través de Internet por las mismas razones que los artículos de robótica manifiestan, la velocidad y el rendimiento no es la necesaria para la velocidad requerida en la transferencia de datos.


antenna
rest of the world
building
hydraulic
azimuth
altitude
mirror 1
mirror 2
mirror 3
rotatorA+B
adapterA+B
guidingA+B
image an A+B
tcs workstation
GPS receiver
time bus distribution
router
science CCDs
emmi control
calibration
science CCD
susi control
tccd worstation
emmi/susi ws
time server
archive ws
pipeline ws
X terminals
Telescope Control LAN
Guide Acquisition LAN
Instrument LAN's
General Services LAN
router


Ethernet
ATM
1 per Telescope
Telescope Control LCUs
Telescope Control LAN
M2 unit LCU
Rapid guiding LAN
Guide Camera LCUs NA,NB,Ca,Co
Image analysis LCUs NA,NB,Ca,Co
Guide/ Acquisition LAN
1 per Instrument
Instrument LAN
Instrument Control LCU
Detector LCU
Switch
Telescope Area
Control Room
Instrument WS
Telescope Control WS
Guide/ Acq. WS
1 for entire Observatory
ASM LAN
ASM LCU
To: Rest of the world
ASM WS
Switch/ Router
Switch/ Router
Backbone LAN
To: UT2, UT3, UT4, Combined Focus, VLTI
Archive WS
Access & Conf Ctrl WS
General Services LAN
Data Handling Server
User WS
User Station LAN

Como se explico anteriormente, NTT sería utilizado como un caso de estudio para VLT Software, y VLT Software consiste en un sistema de computación completamente distribuido [58], [59] y es por la misma razón que se propuso probar en NTT dadas las características distribuidas del mismo. El Observatorio Paranal en sus inicios poseía una disposición lógica como la mostrada en la ilustración 26. Actualmente las redes ATM han desaparecido y todo está en función de fibra óptica y redes ethernet sobre protocolo TCP/IP. VLT Software, para el año 1997, contando comentarios y lineas de código del Software de Control poseía cerca de 1 millón de lineas de código, de las cuales las capas de control y/o capas inferiores están desarrolladas en C y las capas superiores o de la IWS en C++ mientras que las interfaces gráficas, fueron desarrolladas usando el VLT Panel editor (desarrollo de la ESO y el VLT SW Group) basadas en Tcl/Tk. Una de las diferencias y ventajas de VLT Software, es la alta integración entre los sistemas de control y los sistemas de flujo de datos (Data Flow System), lo cual lo diferencia de otros sistemas de control y lo hace más específico para la astronomía y también más particular, es decir, limita su campo de uso a lugares similares a Paranal (lo cual es muy difícil).

Más adelante cuando el software entró en producción, se propuso un nueva arquitectura de funcionamiento (y también funciona como framework) llamada Base Observation Software Stub (BOSS) [60], [61], el cual consiste básicamente en plantillas para el control de instrumentos, de esta forma los desarrolladores sólo debían modificar partes puntuales de un instrumento sin tener que desarrollar, modificar o integrar todo al software, sólo lo necesario. Así también existen los Observation Software (OS) y los SuperOS. Lo que significa que un OS controla otros OS. BOSS se caracteriza por dividir el desarrollo de software de instrumentos en 3 partes:

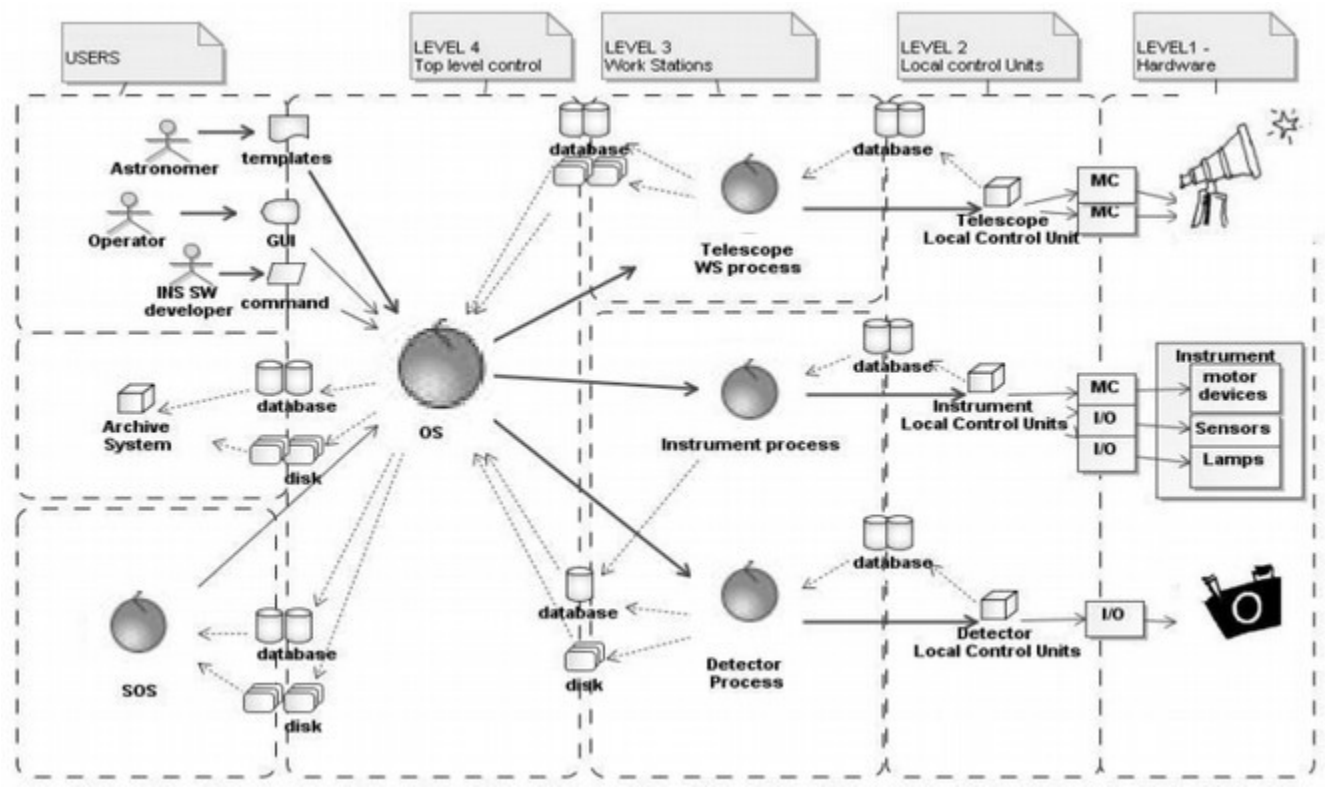
1. Una parte independiente del instrumento que incorpora las propiedades comunes a todo el software de observación.
2. Una parte dependiente del instrumento que es común para un subconjunto de instrumentos
3. Una parte específica del instrumento que caracteriza el funcionamiento del instrumento

BOSS creció casi el doble cuando se comenzó con el proceso de generalización, sin embargo, produjo grandes ventajas a la hora de desarrollar software para nuevos instrumentos y lograr su integración, esto se puede visualizar en la ilustración 27, en la cual es posible ver que las lineas de código por instrumento son bajas en comparación con la cantidad de lineas que posee BOSS, de esta forma el desarrollo y la mantención de instrumentos se hace mas sencilla. Otra ventaja de BOSS es que las actualizaciones y soluciones de problemas de BOSS son transversales a todos los instrumentos lo cual representa una gran ventaja a la hora de resolver problemas que afectan a todos los instrumentos. Sin embargo, VLT Software sigue siendo un software complejo y su curva de aprendizaje es muy empinada [62].

| *OS* | *Size*[1] | *Year*[2] | *Comment* | *OS* | *Size* | *Year* | *Comment* |
|---|---|---|---|---|---|---|---|
| **TestCam** | 0 | 1998 (port ~2002) | VLT instrument (outdated) | **SUSI**<br>**EMMI**<br>**HARPS**<br>**GRONT** | 106<br>130<br>92<br>113 | 1998 (port ~2001)<br>1998 (port ~2004)<br>2003<br>2007 | La Silla Instruments |
| **MIDI**<br>**VINCI**<br>**ARAL**<br>**AMBER** | 0<br>179<br>150<br>6219 | 2002<br>2000<br>2004<br>2004 | VLTI instruments | | | | |
| **SINFONI**<br>**SPIFFI**<br>**MACS** | 7432<br>556<br>81 | 2004 | VLT instrument; SOS using adaptive optics | **FLAMES**<br>**GIRAFFE**<br>**FP** | 1946<br>193<br>511 | 2002 | VLT instrument; SOS |
| **CRIRES** | 2560 | 2006 | Spectrograph, Adaptive optics | **HAWKI** | 722 | 2007 | Image taking camera |
| **VISTA** | 3197 | 2008 | VISTA instrument | | | | |
| UVES | 14286 | 1999 | Non BOSS based OS | | | | |
| ISF | 11674 | 1999 | NAOS/CONICA base (main module)– starting point of BOSS | | | | |
| **BOSS** | 25541 | 2008 | Base Observation Software (main module) | | | | |

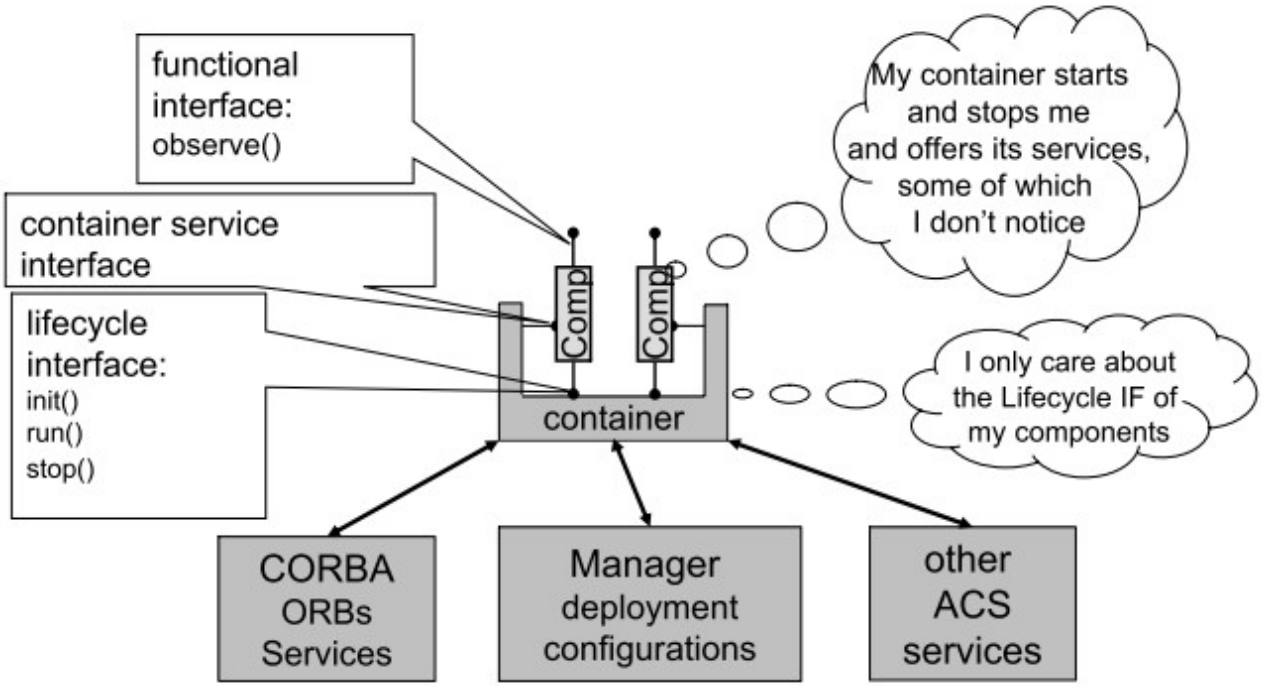
*Ilustración 27: Tamaña de OS y BOSS por instrumento, fuente [60]*

En la ilustración 28 se pueden visualizar los diferentes procesos que interactúan al realizar una observación



El creciente desarrollo de las tecnologías industriales, principalmente Programable Logic Controller (PLC), provoca que varias empresas que proveen de materiales o repuestos para los actuales (y antiguos) sistemas no los sigan desarrollando, un caso puntual es el de las tarjetas de control de movimiento MACCOM para VME Bus, la empresa que las proveía, dejó de desarrollarlas por lo que ESO tuvo que invertir en el desarrollo y/o reparación de las defectuosas [62]. Uno de los últimos esfuerzos realizados con el instrumento Pioneer en Paranal, pretende dar las luces de la factibilidad de migrar la plataforma de control sin intervenir el software de observación, de esta manera se afecta solo la parte de control, utilizando hardware de la industria. Siguiendo esta línea, no solo VLT propone trabajar con componentes industriales. En [63] se puede apreciar otro observatorio que utiliza en la parte superior, un software de Supervisión, Control y Adquisición de Datos (SCADA) desarrollado en Python sobre Linux, los sistemas de bajo nivel son ejecutados por controladores industriales. En la actualidad, muchos de los nuevos instrumentos han adoptado la opción de componentes del ámbito industrial y se avanza en la migración a controladores industriales PLC Beckhoff [64].

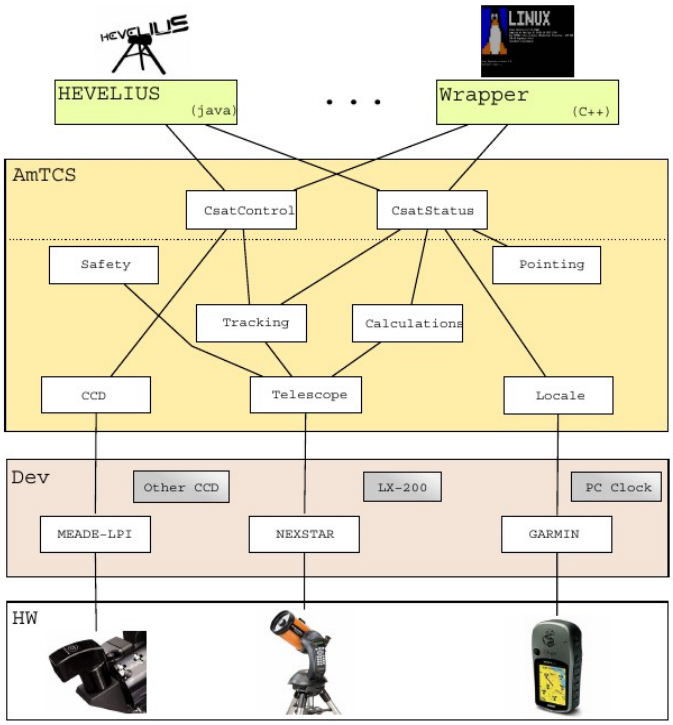
Por otra parte, el observatorio radiostronómico ALMA también posee una arquitectura de software basada en software distribuido, de hecho, deben lidiar con equipos que están a 0.5km o 14km sin contar equipos que reciben datos desde Santiago o Europa cuyas interfaces de software pertenecen al framework de ALMA llamado ACS (ALMA Common Software) [65], [66]. ACS a diferencia del software desarrollado para VLT, está basado en CORBA y ha sido cuidadosamente diseñado para ser independiente de algún software comercial, por ejemplo, usando Objects Requests Brokers como TAO y JacORB. Además, ha sido cuidadosamente diseñado para minimizar la duplicación de funcionalidades y maximizar la eficiencia del framework y tratando de abstraer completamente los detalles técnicos/mecánicos de los requerimientos científicos haciéndolo un software transparente para los astrónomos que no poseen un acabado conocimiento de radioastronomía. Otra de las características de ACS es su modelo de container/component lo que significa que el SW debe ser dividido en módulos que puedan ser desarrollados por grupos individuales y puedan ser integrados a un sistema coherente. Este modelo de container/component es una característica fundamental del framework. En la ilustración 29 se puede apreciar un esquema gráfico de dicha característica.

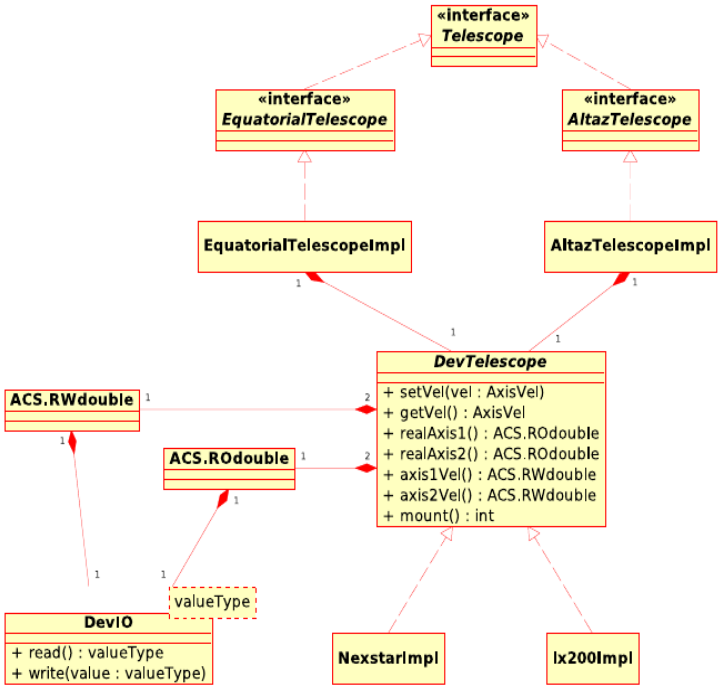



Sin embargo, ACS está pensando para ser utilizado en ALMA y para satisfacer los requerimientos de ALMA[67], por lo mismo, tal vez no es la mejor solución para observatorios que cuentan con un telescopio, una o dos cámaras y nada más. Sin embargo, la arquitectura y el modelo propuesto son buenas opciones para desarrollar o adaptar software más sencillo con el concepto de container/component. De esta forma, se lograrían los mismos beneficios que ha logrado ACS pero con un software que sería más pequeño y más sencillo de mantener por un grupo pequeño de desarrolladores [20].

A pesar de que ACS es un framework complejo y robusto, han habido intentos de utilizar este software para el control de telescopios amateurs, por ejemplo en [12] se muestra un intento de un grupo de estudiantes de la Universidad Federico Santa María, que intentan generar un TCS (Telescope Control System) utilizando como framework de desarrollo ACS. Sin embargo, la principal traba que encontró este equipo de investigación fue definir interfaces genéricas que puedan representar una variedad de dispositivos (ver ilustraciones 30 y 31). Siguiendo el enfoque de modularidad de ACS, se desarrolló el TCS pensando en trabajar con cualquier tipo de montura de telescopio, particularmente monturas Altazimutal y Ecuatorial sin que esto influya en la forma con que el telescopio se opera. Es decir, siguiendo la misma lógica de abstracción de ACS, se brinda una interfaz para operar un telescopio independiente de que el astrónomo sepa o no usar una montura Altazimutal o Ecuatorial, con el objetivo de indicar al telescopio qué observar o donde enfocar delegando al sistema como resolverlo dependiendo de la montura que se esté utilizando. El problema es que este software funcionó sólo en 2

de 3 telescopios/monturas probadas, según los autores, demostrar que se pueden operar estos 3 tipos de telescopios es una clara evidencia de que ACS podría ser utilizado para generalizar el control de telescopios construyendo los drivers apropiados para cada tipo de telescopio.



## Software de pequeños observatorio

Luego de discutir el software para dos de los observatorios más grandes del mundo y que además son de diferentes funcionalidades (óptico y radioastronómico), los siguientes párrafos serán dedicados a analizar sistemas e infraestructuras de pequeños observatorios, de los cuales, cada uno ha hecho su propia implementación.

Por ejemplo en [68] se muestra un framework para controlar un telescopio óptico ubicado en Mauna Kea, Hawaii, operado por NAOJ (National Astronomical Observatory of Japan). En este artículo se muestra un sistema distribuido construido especialmente para este telescopio y que intenta satisfacer los mismos requerimientos de ACS y VLT Software, agendar observaciones en una plantilla para luego enviarlas el OCS y que éste pueda ir descomponiendo la plantilla hasta lograr operaciones de bajo nivel para cada instrumento o subsistema. El software de control fue desarrollado completamente en Python, puesto que tanto para ingenieros como para astrónomos, es mas fácil desarrollar nuevas funcionalidades o realizar correcciones. Además Python es un lenguaje que a ganado una gran importancia en la comunidad científica dada su facilidad y la gran cantidad de herramientas disponibles para la ciencia.

Por otra parte, en [69], [70] se habla sobre la importancia de la observación remota y como esta puede reducir los costos de operación, en este trabajo se muestra como un observatorio operado con sistemas manuales y de electrónica local, pudo ser migrado a un sistema controlado computacionalmente para luego realizar observaciones de forma remota, esto gracias a los avances en automatización y sistemas de control computacionales. Pero, al igual como se reportaba en los primeros trabajos de telerobótica, el ancho de banda fue un gran problema, esto porque el lugar se encontraba lejos del alcance de la conectividad. El software es claramente un sistema distribuido desarrollado principalmente con KTL para los servicios y Tcl/Tk (el mismo software utilizado para desarrollar interfaces en VLTSW) en el desarrollo de las interfaces gráficas. La conexión remota fue implementada con VNC sobre SSH, lo que además implica que el sistema operativo utilizado fue Linux. Por otro lado, a diferencia de los otros software revisados, el control del domo y los sistemas de alarmas, fueron implementados después de tener todos los sistemas funcionando y en operación. Finalmente se rescata la importancia de realizar chequeos físicos al momento de la implementación, esto porque hay algunos detalles que podrían no ser notados durante la operación remota, hay que tomar en cuenta que hay muchos factores que pueden afectar al observatorio y todos deben estar sensados, lo que requiere de mucha planificación pero sobre todo, de una completa verificación al momento de implementar la solución. La mayoría de los trabajos revisados recomienda una amplia fase de implementación.

Siguiendo con observatorios remotos, en [71] se propone un proyecto ambicioso que busca generar una red de observatorios para observar diferentes fenómenos en diferentes lugares del mundo, esto corresponde a un sistema completamente distribuido donde cada nodo del sistema corresponde a un observatorio que es controlado por un computador. A su vez, cada nodo es un observatorio diferente, que no necesariamente fue construido con las mismas especificaciones, por lo que si bien es cierto se

comunica por el lado del framework a través de una interfaz, cada nodo ve cómo lo hace para publicar la información solicitada. Un dato curioso acerca del funcionamiento de esta red de observatorios, es que no se hacen mantenciones preventivas, los problemas se resuelven tan rápido como se presentan, esto para evitar perder noches de observación en trabajos técnicos.

Respecto de las fallas detectadas en la implementación de esta red, indican que el escenario que tratan de prevenir con mayor énfasis es el de dejar los instrumentos expuestos cuando hay mal clima, nada mós puede causar tanto daño como lluvia dentro del observatorio porque algo falló al cerrarse. Uno de los problemas reportados en el artículo, fue que un operador olvidó cerrar el domo al final de la noche, al día siguiente una lluvia alcanzó el observatorio y mojó los instrumentos y la electrónica, afortunadamente sólo la tarjeta de la cámara CCD resultó dañada y esta fue reparada por el fabricante, lo malo es que el observatorio estuvo cerrado por 2 meses antes de volver a funcionar. Otro problema reportado es la energía con que los observatorios funcionan, en un observatorio hubo un corte eléctrico y la UPS no alcanzó para lograr cerrar el domo, hubo una lluvia y todos los instrumentos y la electrónica se mojó, por lo que se dejó secando y el observatorio estuvo operativo después de una semana. Con esta lección la mejor recomendación obtenida de este artículo es que los sensores, todos deben estar correctamente integrados al sistema de control para que estos cumplan con su función y el software de control reaccionar de forma segura con la información que está recibiendo de los sensores. Además, dicho software de control debe ser capaz de recuperarse automáticamente luego de una falla de sistema. Otro de los problemas reportados fue que el sistema se detuvo y estuvo toda una noche en la que se esperaba generar datos sin obtenerlos.

En [72] se puede ver otro esfuerzo por realizar un framework para una red de observatorios utilizando la tecnología del Colisionador de Hadrones. El estudio consiste en un framework llamado GLORIA (GLObal Robotic-telescope Intelligent Array). El estudio se basa en la utilización del framework de reducción de datos del LHC para fines astronómicos, según su estudio el software utilizado en el LHC, este podría procesar datos equivalentes a tomar una imagen de 100 mega píxeles cada 50 nanosegundos. Finalmente el artículo indica como se pueden hacer descubrimientos con observatorios remotos usando un software distribuido de alto rendimiento para detección de estos.

En [13] se muestra otro esfuerzo por lograr la robotización de un observatorio, llama profundamente la atención que a la fecha de publicación (2010) ya varios esfuerzos y documentos realizados por otros investigadores habían sido publicados para realizar el control de un observatorio remoto, sin embargo en esta universidad se optó por un desarrollo propio. De este estudio podemos destacar algunos resultados importantes, como por ejemplo la precisión del autoguider (que es en pocas palabras el software de control que permite seguir una estrella) y el hecho de que es posible desarrollar software e implementar soluciones de control en las Universidades tal y como lo describen en este artículo.

Llama además la atención que la mayoría de los artículo revisados poseen una estructura de funcionamiento similar [15], en términos generales, poseen un OT (Observation Tool) que se encarga de administrar la observación, el cual recibe una plantilla desarrollada por el astrónomo, que dirá como

se debe comportar el observatorio, el telescopio o los instrumentos, luego el software automáticamente divide la plantilla en pasos cada vez más pequeños que los diferentes subsistemas puedan entender hasta llegar a las instrucciones de más bajo nivel que son las que deben entender los instrumentos y la maquinaria en general.

Otro trabajo interesante de revisar, es un artículo desarrollado en la Universidad Católica de Chile, a cargo del Centro de Astro-ingeniería de dicha Universidad [20]. En este artículo, un análisis importante es hecho en función de qué software utilizar para desarrollar el sistema de control de un observatorio de 50cm con montura ecuatorial que pertenecía al observatorio La Silla y que luego fue donado al observatorio Santa Martina (perteneciente a la UC). En este trabajo se analizan software como VLTSW, ACS, TANGO, EPICS y ICE, de los cuales se elige ICE dado que provee todas las funcionalidades de objeto distribuido como CORBA pero mucho mas liviano. La principal razón por la que se elije ICE, es porque los otros software son demasiado complejos para un equipo pequeño de desarrollo. Además varios de los participantes de este proyecto eran personas que ya habían trabajado con VLTSW y ACS y otros software basados en CORBA y dan fe de que no olvidaron ninguna característica. Se escogió ICE porque es mucho más sencillo de desarrollar. Por otra parte en [73] se muestra un software desarrollado para observatorios solares llamado ATST que presenta un modelo componente/contenedor similar al de ACS para usarlo en un instrumento del ATST. El esquema de funcionamiento es muy similar a los propuestos por los otros observatorios, sin embargo el software no es el mismo, llama bastante la atención que teniendo ideas similares, cada centro tiende a desarrollar su propio framework para luego tratar de reutilizarlo en otros lugares o en otras situaciones. En [74] se muestra otro ejemplo de un observatorio con computación distribuida, pero en este caso sobre 2 telescopios situados en el mismo lugar para funcionar como uno, la ventaja es que pueden usar técnicamente dos instrumentos simultáneamente. De este trabajo se destaca la importancia que toma el sistema de control respecto del clima, donde las observaciones son decididas en función del clima. Otro tópico importante, es realizar una programación de observaciones que optimice el uso del telescopio y los instrumentos. Hay observaciones que tal vez no puedan ser realizadas con determinadas condiciones lumínicas o atmosféricas, por lo tanto lograr una correcta programación, conduce a optimizar el tiempo de observación del telescopio lo cual también es analizado en [16], [75] que por lo demás es otro software diseñado para computación distribuida usando CORBA.

## Software Libre para Astronomía

Para finalizar la revisión de software ligados a la astronomía, se discutirán algunos software OpenSource disponibles en Internet para ser utilizados en el control de telescopios, domos y cámaras. Estos software son: ASCOM [76], TheSkyX [77], INDI Library [78], Stellarium [79] y RTS2 [80].

El objetivo principal de ASCOM [81] es proveer un nivel de driver-cliente para separar diversos tipos de dispositivos astronómicos del software que utilizan y proveer las API para diversos tipos de lenguajes de programación, de esta forma, cada desarrollador o investigador puede escoger el lenguaje que más le acomode para construir sus aplicaciones.

Antes de ASCOM (en términos de software de astronomía amateur) habían dos formas de hacer sistemas de control para instrumentos astronómicos, el primero, una arquitectura simple y el segundo una arquitectura extensible. Para la arquitectura simple, todo el control de los dispositivos queda en un solo paquete, si se quiere cambiar o agregar un nuevo dispositivo, el paquete completo debe ser re-escrito, si un problema aparece en el sistema de control, todo el paquete debe ser actualizado, lo que convierte a este sistema en una arquitectura de relación uno-a-uno entre el equipo y el software. Con la aparición del concepto escalable, la relación se transformó en uno-a-muchos, puesto que ahora había una capa de plug-in, a la cual se le podían incorporar diversos dispositivos. En otras palabras cuando un nuevo instrumento aparecía, sólo había que desarrollar el driver para ese equipo e integrarlo al sistema sin cambiar el resto del software. Si un problema aparecía en el dispositivo, solo se debía arreglar el paquete relacionado con el dispositivo y no todo el sistema. Con la llegada de ASCOM, la relación se tornó muchos-a-muchos, ahora se podía tener diversos tipos de software utilizando diversos tipos de hardware y ambos conectados por un factor común, ASCOM. De esta forma, fue posible desarrollar diversos sistemas de control sin importar que dispositivo se fuese a conectar siempre y cuando este dispositivo cumpliera con el estándar y tuviese un driver ASCOM. Por el lado del dispositivo, los fabricantes se encargaron de fabricar drivers ASCOM para sus dispositivos, por lo tanto, cualquier software se podría conectar con cualquier dispositivo siempre y cuando este hardware tuviese su driver ASCOM. De esta forma, es mucho mas sencillo desarrollar y compartir software para diversos dispositivos sin tener en mente un dispositivo particular. ASCOM es una capa de drivers desarrollada para Windows usando el concepto de COM (Component Object Model).

Por lo tanto, a ojos de la comunidad astronómica, ASCOM representó un gran progreso puesto que puso en manos de desarrolladores, la integración de diversos sistemas e introdujo el concepto “plug and play” en equipos astronómicos. El problema principal que presenta ASCOM es la poca cantidad de desarrolladores que podrían, o están desarrollando aplicaciones basadas en ASCOM, por lo tanto conseguir soporte o conseguir a un programador que pueda integrar equipos nuevos, desarrollar drivers o nuevas funcionalidades es bastante complicado.

Otro software bastante popular dentro de la astronomía amateur es TheSkyX [77], pero posee una arquitectura de tipo uno-a-uno, lo que significa que si bien el software es muy robusto y completo,

pierde dinamismo al depender de un hardware particular para funcionar. El hardware y el software están fuertemente ligados y es muy complejo separarlos. Varios dispositivos no pueden ser controlados con este sistema de control porque los desarrolladores no los han incorporado y no existe la posibilidad de agregar más dispositivos, por lo tanto, la reusabilidad de código es bastante pobre. Sin embargo, es bastante robusto y eficiente para los dispositivos para los cuales fue diseñado. En el ámbito de la investigación, donde siempre se están desarrollando nuevos instrumentos, contar con este tipo de restricciones es determinante a la hora de decidir el software de control a utilizar para un proyecto. TheSkyX cuenta con una serie de dispositivos con los cuales es compatible, sin embargo sigue siendo un paquete de software distribuido para ser compatible solo con dichos dispositivos, por lo tanto la relación uno-a-uno es permanente, para agregar compatibilidad con nuevo hardware, es necesario generar una nueva versión del software, esto ligado al hecho de que es un software pagado, por lo tanto no está entre sus objetivos compartir software si no mas bien lograr que clientes compren su software y bajo esa misma filosofía, no es conveniente que otros desarrolladores intervengan o sepan como funciona el software. En resumen y en función de lo que antes se ha discutido, si bien es cierto TheSkyX es un software potente, está enfocado en proyectos en los que se tenga poco tiempo de integración o poco capital humano para desarrollar la integración y se compra una solución completa para la rápida puesta en marcha, sin embargo, en ambientes de innovación continua, de crecimiento continuo, este tipo de soluciones es bastante poco viable, principalmente porque es imposible desarrollar compatibilidad con nuevos equipos.

Luego se encuentra INDI Library (Instrument Neutral Distributed Interface) [78], que consiste en un proyecto Open Source para el control de diversos tipos de equipos astronómicos (domo, cámara CCD, telescopios, video, por nombrar algunos), y su arquitectura es de tipo cliente-servidor. Similar a la propuesta de ASCOM, INDI pone entre el cliente y los dispositivos un servidor, llamado indiserver. Este servidor hace de intermediario entre el cliente y cada uno de los drivers, de esta forma permite tener un control local y remoto de los dispositivos, es fácil construir nuevos drivers para integrarlos a INDI y crear aplicaciones que puedan interactuar con un dispositivo. Por lo tanto, se puede decir que tiene una arquitectura muchos-a-muchos, y, como es un servidor, este puede tener varios clientes (varios drivers). Por tanto diversos clientes pueden controlar diversos equipos. La comunicación entre cliente y servidor es a través de comunicación TCP vía sockets. Este software también es compatible con otras aplicaciones como Kstars, con lo que además de controlar el telescopio o la cámara usando INDI, se puede utilizar una Interfaz Gráfica con un mapa estelar para indicar donde el telescopio debería apuntar. El mayor problema que tiene este software es el soporte, no es ampliamente difundido y está completamente basado en C y C++, no permite desarrollo en otros lenguajes, aunque si tiene un protocolo de comunicación basado en XML para enviar ordenes, por lo tanto, se podría desarrollar un cliente que genere ordenes para interactuar con INDI basado en XML.

Stellarium [79] es otro software popular utilizado en astronomía para visualizar mapas estelares con cielo realista y en 3D, dentro de sus opciones está incorporado el control de telescopios, por lo tanto un usuario podría marcar una estrella en el mapa virtual y luego indicar al telescopio que apunta ese

cuerpo celeste. Si bien es cierto, este es más un software de visualización que un software de control, se destaca como una opción dada su popularidad dentro de la comunidad científica y amateur. Cuenta con varias versiones y un gran equipo de desarrollo  y además está disponible en los repositorios de Ubuntu.

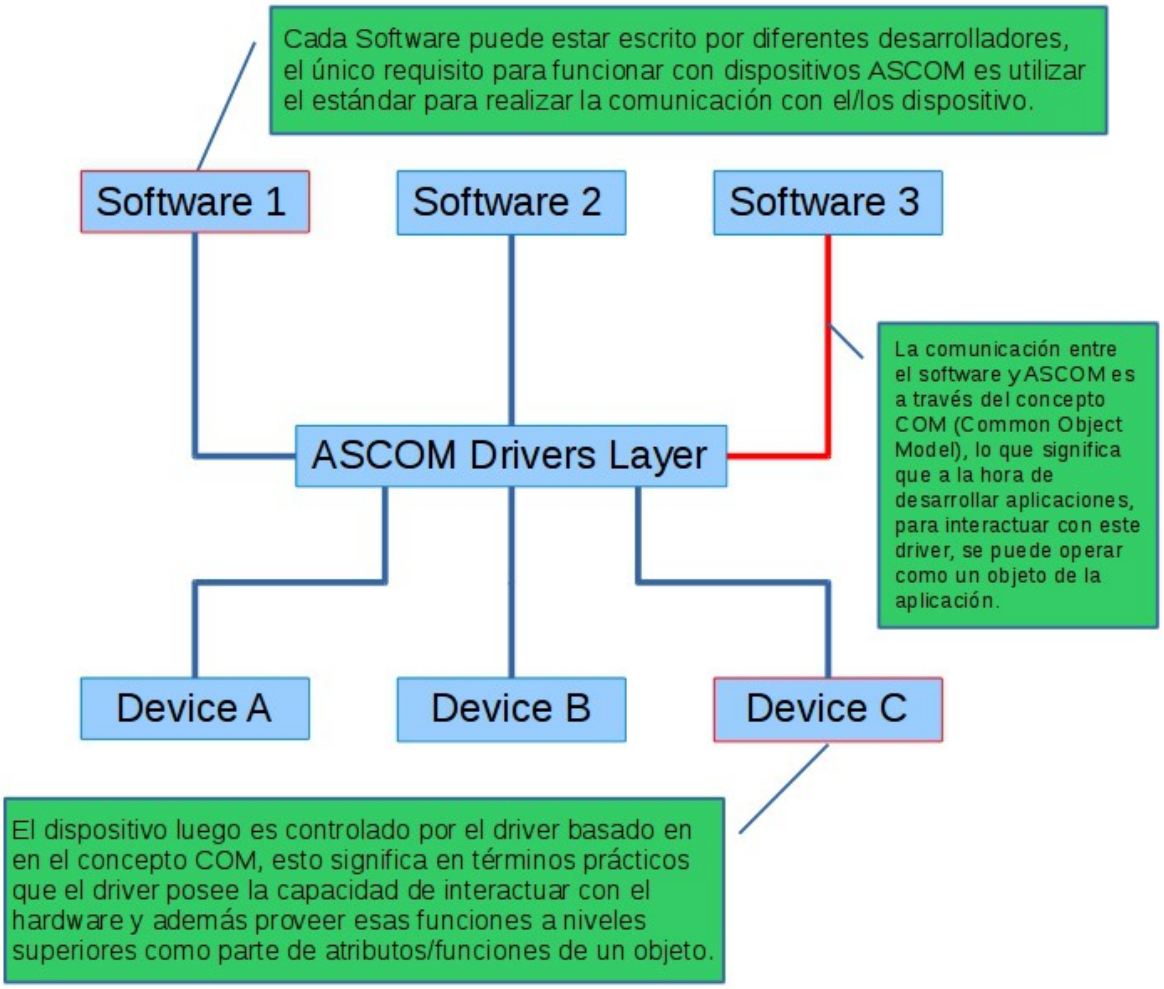
Por último, se encuentra RTS2 (Remote Telescope System 2) [80], que posee un objetivo similar al presentado por INDI y funciona principalmente bajo Linux. Sin embargo, este software está en funcionamiento (según su pagina) en alrededor de una docena de observatorio completamente teleoperados. Su diseño modular permite fácilmente agregar nuevos dispositivos. Este software además tiene la posibilidad de enviar información en formato INDI, lo que da la posibilidad de leer información usando Kstars entre otros. De forma general, RTS2 es similar a INDI, tiene un esquema o arquitectura similar a ASCOM y por ende son software muy parecidos, sin embargo, RTS2 está implementado a nivel mundial en muchos lugares [80]. Además, es posible encontrar artículos científicos publicados relacionados con RTS2 por lo tanto hay un avance e interés en el desarrollo de esta plataforma [71], [80], [82]–[87].

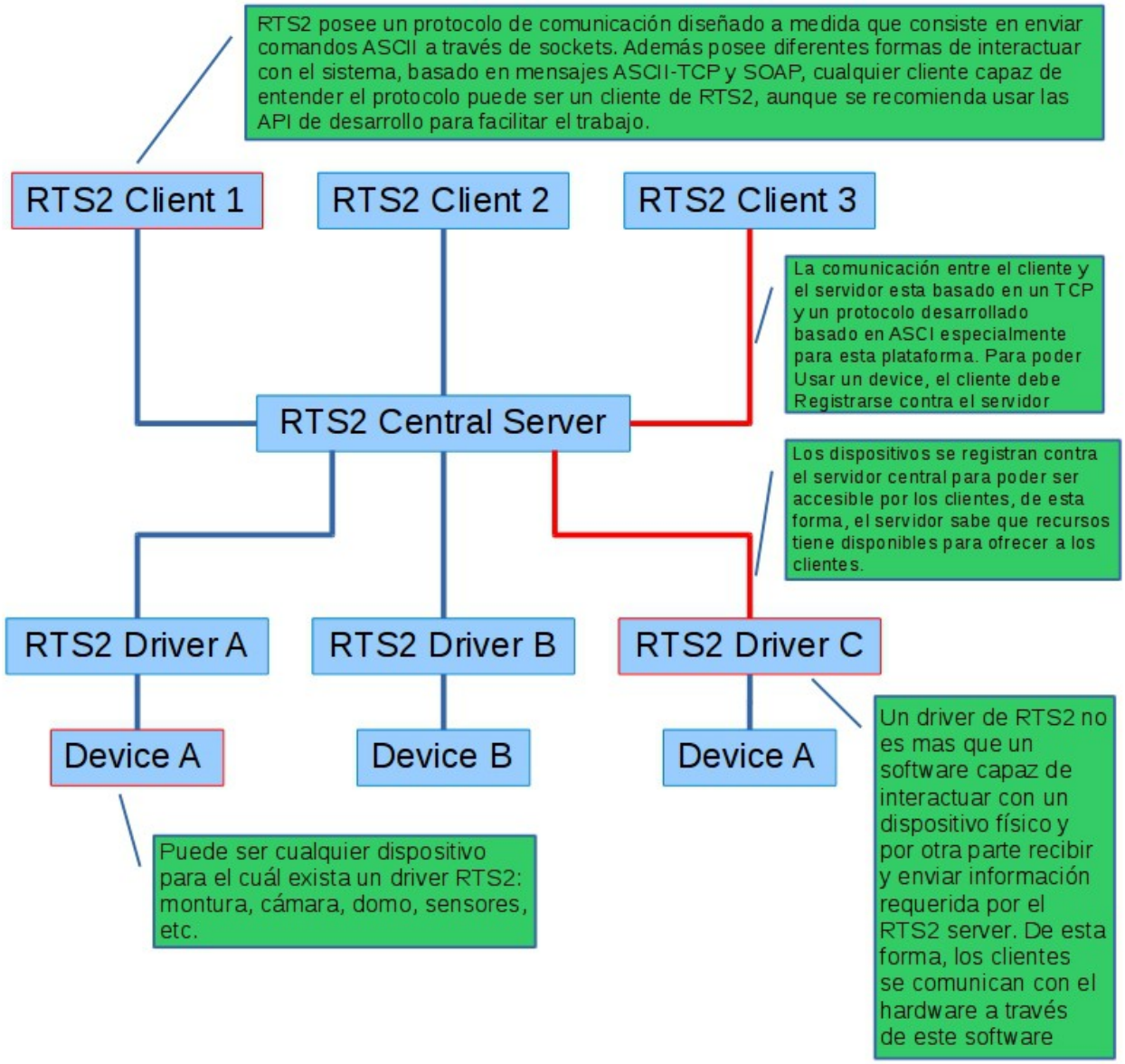
Del estudio realizado en este capítulo, se puede concluir que varios de los software analizados funcionan de forma similar. Se realizó una búsqueda en amplitud acerca del funcionamiento de los observatorios hoy en día y la tendencia es clara, la computación distribuida, la teleoperación, los observatorio remotos y los observatorios robóticos son el nuevo norte en la astronomía, principalmente porque los observatorios deben estar alejados de la contaminación lumínica de las ciudades y en lugares que generalmente son inhóspitos,  alturas y temperaturas extremas.

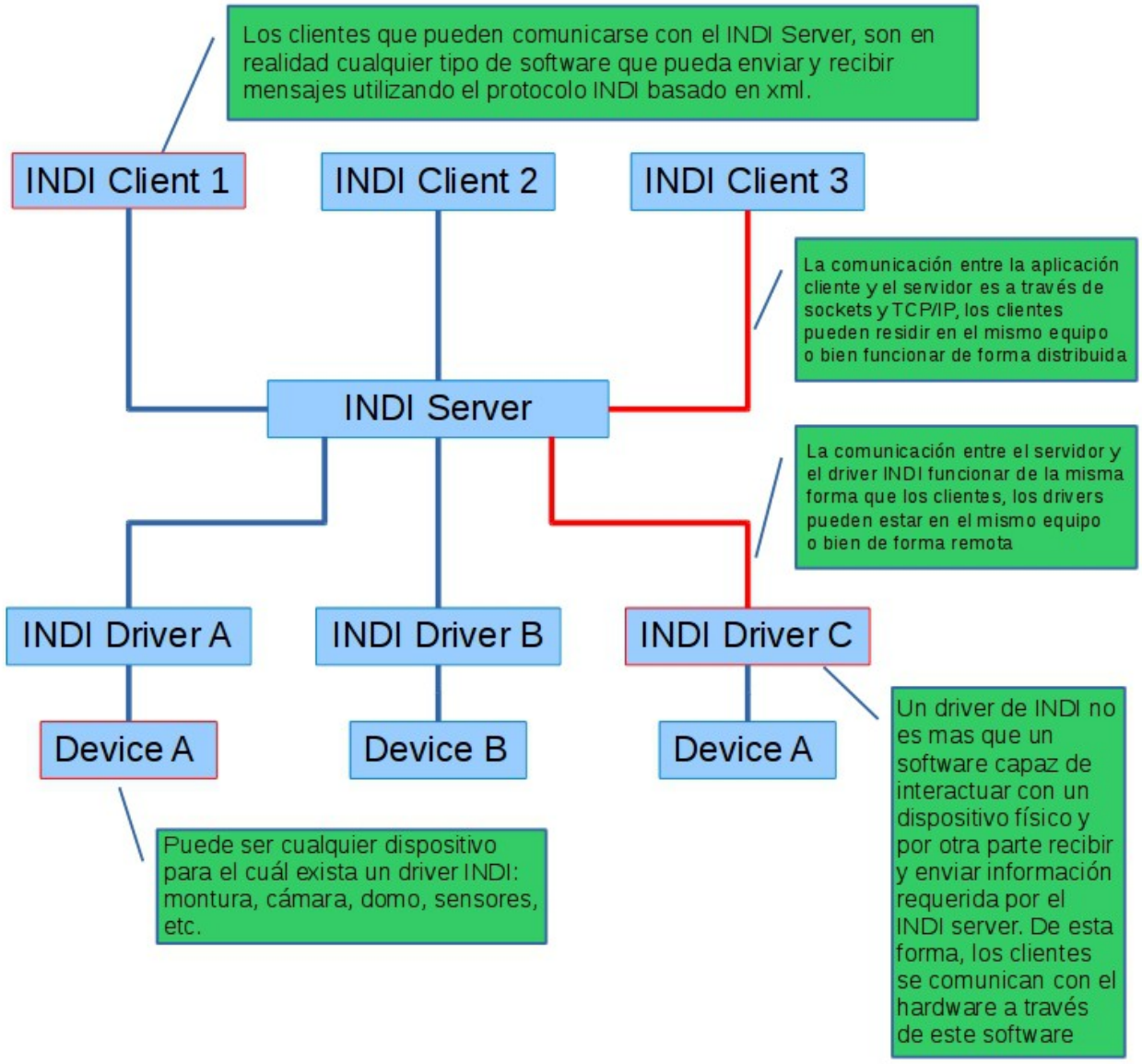
Dentro de los factores comunes encontrados en el análisis del estado del arte, el uso de CORBA como middleware fue común en la mayoría de los artículos así como también el uso del sistema operativo Linux o Unix. Por otra parte, muchos estudios y desarrollos tratan de resolver problemas similares, una y otra vez, desarrollando sus propios software y aplicaciones diseñados a medida, con lo cual se espera tener un alto rendimiento pero a costa de un prolongado tiempo de desarrollo y testing. Como se puede apreciar además en [14], hoy en día el desarrollo de software está pensado para ser mantenible a través del tiempo, para poder ser integrado como parte de otros sistemas o permitir al mismo sistema, anexar nuevas características y además realizar desarrollos portables. Por otra parte, como se discute en [1] la comunidad astronómica es reducida y ampliamente especializada, por lo tanto contar con una comunidad pequeña y que a su vez esta comunidad genere subcomunidades para generar subproductos para la misma comunidad no tiene mucho sentido. Es importante que el software desarrollado para astronomía, pueda ser desarrollado como parte de otro software que no necesariamente sea mantenido por esta pequeña comunidad, un ejemplo claro son VLTSW y ACS, donde sólo quienes trabajan para ESO (Paranal y La Silla principalmente) y ALMA mantienen su software, con lo cual la posibilidad de realizar mejoras o integración de nuevas características es un tema complejo. Estos frameworks requieren de mucho tiempo de aprendizaje, cabe destacar que entender estos grandes frameworks es una tarea compleja, por lo mismo, encontrar gente que pueda trabajar en estos sistemas es aún más complicado. Por otra parte, utilizar software libre, especializado en astronomía, es aún mas complejo

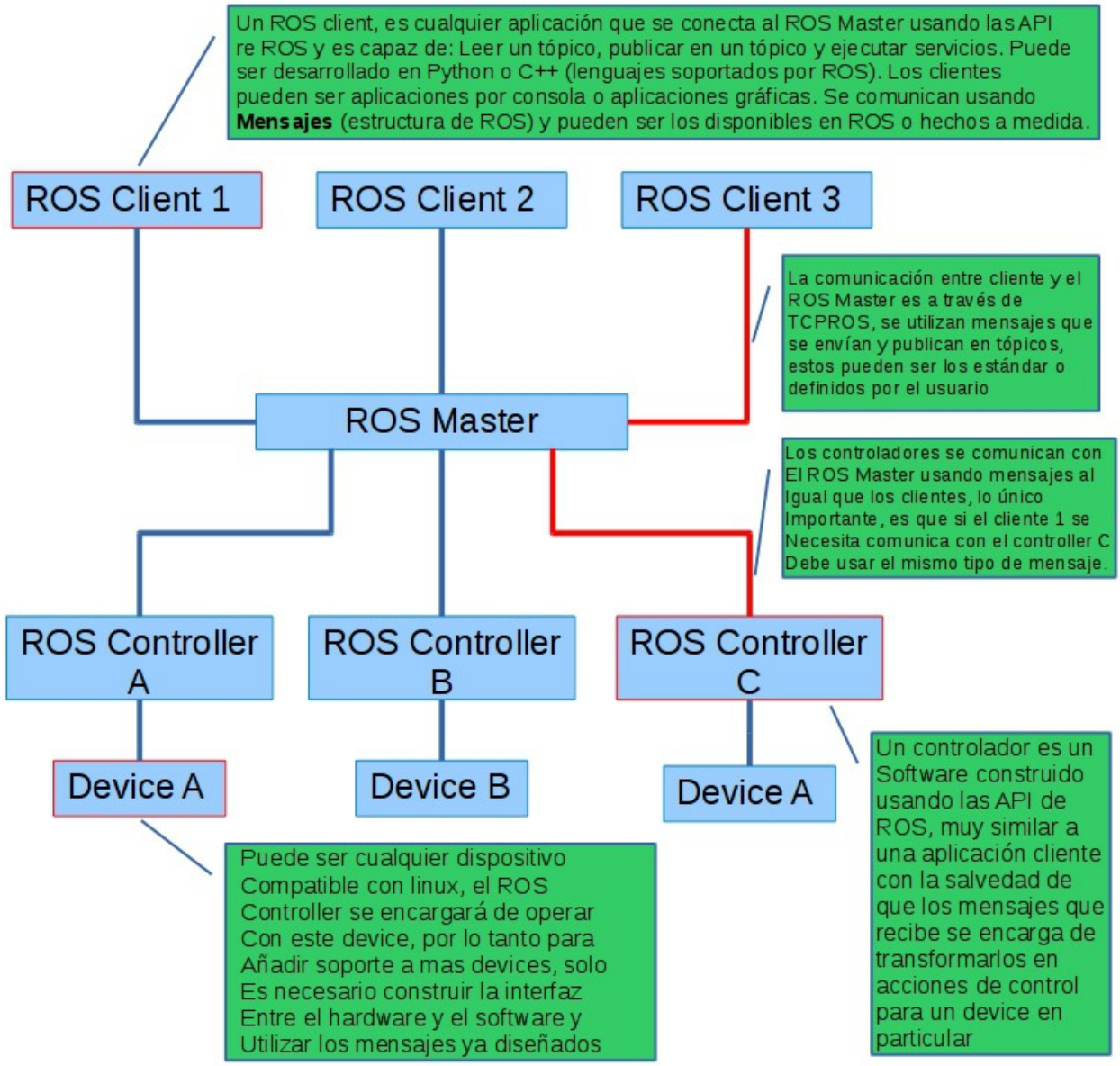
porque ni siquiera se asegura un nivel mínimo de soporte, con lo cuál, optar por software ampliamente utilizado en otro ámbito para luego adaptarlo de alguna forma a su uso en astronomía pareciera ser la mejor opción a la hora de conseguir soporte y asegurar la sustentabilidad del sistema.

Los siguientes diagramas, son resultado del análisis de diferentes software y el funcionamiento general de la mayoría de los softwares analizados, se tomarán como referencia: ASCOM (32), RTS2 (33), INDI (34) y luego la solución propuesta usando ROS (35).




RTS2 posee un protocolo de comunicación diseñado a medida que consiste en enviar comandos ASCII a través de sockets. Además posee diferentes formas de interactuar con el sistema, basado en mensajes ASCII-TCP y SOAP, cualquier cliente capaz de entender el protocolo puede ser un cliente de RTS2, aunque se recomienda usar las API de desarrollo para facilitar el trabajo.
RTS2 Client 1
RTS2 Client 2
RTS2 Client 3
La comunicación entre el cliente y el servidor esta basado en un TCP y un protocolo desarrollado basado en ASCI especialmente para esta plataforma. Para poder Usar un device, el cliente debe Registrarse contra el servidor
RTS2 Central Server
Los dispositivos se registran contra el servidor central para poder ser accesible por los clientes, de esta forma, el servidor sabe que recursos tiene disponibles para ofrecer a los clientes.
RTS2 Driver A
RTS2 Driver B
RTS2 Driver C
Device A
Device B
Device A
Un driver de RTS2 no es mas que un software capaz de interactuar con un dispositivo físico y por otra parte recibir y enviar información requerida por el RTS2 server. De esta forma, los clientes se comunican con el hardware a través de este software
Puede ser cualquier dispositivo para el cuál exista un driver RTS2: montura, cámara, domo, sensores, etc.


Los clientes que pueden comunicarse con el INDI Server, son en realidad cualquier tipo de software que pueda enviar y recibir mensajes utilizando el protocolo INDI basado en xml.
INDI Client 1
INDI Client 2
INDI Client 3
La comunicación entre la aplicación cliente y el servidor es a través de sockets y TCP/IP, los clientes pueden residir en el mismo equipo o bien funcionar de forma distribuida
INDI Server
La comunicación entre el servidor y el driver INDI funcionar de la misma forma que los clientes, los drivers pueden estar en el mismo equipo o bien de forma remota
INDI Driver A
INDI Driver B
INDI Driver C
Device A
Device B
Device A
Un driver de INDI no es mas que un software capaz de interactuar con un dispositivo físico y por otra parte recibir y enviar información requerida por el INDI server. De esta forma, los clientes se comunican con el hardware a través de este software
Puede ser cualquier dispositivo para el cuál exista un driver INDI: montura, cámara, domo, sensores, etc.

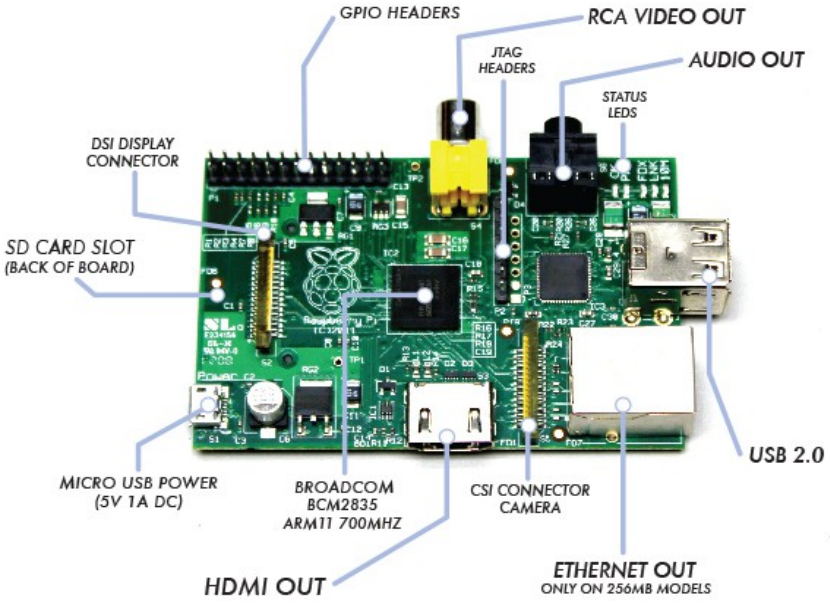

Un ROS client, es cualquier aplicación que se conecta al ROS Master usando las API re ROS y es capaz de: Leer un tópico, publicar en un tópico y ejecutar servicios. Puede ser desarrollado en Python o C++ (lenguajes soportados por ROS). Los clientes pueden ser aplicaciones por consola o aplicaciones gráficas. Se comunican usando **Mensajes** (estructura de ROS) y pueden ser los disponibles en ROS o hechos a medida.
ROS Client 1
ROS Client 2
ROS Client 3
La comunicación entre cliente y el ROS Master es a través de TCPROS, se utilizan mensajes que se envían y publican en tópicos, estos pueden ser los estándar o definidos por el usuario
ROS Master
Los controladores se comunican con El ROS Master usando mensajes al Igual que los clientes, lo único Importante, es que si el cliente 1 se Necesita comunica con el controller C Debe usar el mismo tipo de mensaje.
ROS Controller A
ROS Controller B
ROS Controller C
Device A
Device B
Device A
Puede ser cualquier dispositivo Compatible con linux, el ROS Controller se encargará de operar Con este device, por lo tanto para Añadir soporte a mas devices, solo Es necesario construir la interfaz Entre el hardware y el software y Utilizar los mensajes ya diseñados
Un controlador es un Software construido usando las API de ROS, muy similar a una aplicación cliente con la salvedad de que los mensajes que recibe se encarga de transformarlos en acciones de control para un device en particular

# Raspberry PI

En esta sección, se presentan las características técnicas más relevantes de la Raspberry PI (ver 36) en función del estudio realizado, estas características fueron las determinantes a la hora de decidir la plataforma sobre la cual se trabajó. Toda la información contenida en esta sección fue obtenida desde [88], [89].

## Overview

La Raspberry PI es un computador del tamaño de una tarjeta de crédito al que se le puede conectar un teclado y un monitor. Posee salida de video a través de un conector HDMI o señal análoga. Además tiene una serie de pines GPIO (General Purpose Input/Output) con los cuales se puede realizar interacción directa con otros aparatos a través de la comunicación serial. Existen hasta el momento 4 tipos de Raspberry PI disponibles, modelo A/A+ y modelo B/B+, las diferencias principales entre ambas radican en sus componentes y capacidades de hardware, mientras que las modelo A/A+ tiene 256Mb de RAM, 1 puerto USB y no posee conexión Ethernet, la Raspberry modelo B posee 512 Mb de RAM, 2 puertos USB y un puerto Ethernet mientras que la modelo B+, posee mas pines GPIO, 4 puertos USB y menos consumo energético. La raspberry modelo B/B+ mide aproximadamente 85.6mm x 56mm x 21mm y pesa cerca de 45g.



## Especificaciones técnicas

La Raspberry PI contiene un SoC (System on a Chip) Broadcom BCM2835, éste contiene un ARM1176JZFS (seleccionado por los diseñadores de la Raspberry debido a que presenta la mejor relación coste/rendimiento), capaz de procesar números de coma flotante y de alcanzar una velocidad de 700Mhz. Posee una tarjeta de video de 4 GPU (en total capaz de alcanzar un rendimiento de 1Gpixel/s, 1.5Gtexel/s o 24GFLOPs). Puede reproducir videos en calidad BluRay (1080p a 30fps), utilizando el codec de audio H.264 a 40Mbits/s. y puede dibujar gráficos 3D gracias a las librerías de OpenGLES2.0 y OpenVG. Lamentablemente, agregar un reloj de tiempo real, aumenta los costes increíblemente por lo que la Raspberry no cuenta con un reloj de tiempo real. Además, tampoco es posible agregar más memoria RAM, debido a que está todo integrado en el mismo SoC por lo tanto es imposible modificarlo.

Otro aspecto a considerar es que la Raspberry PI está pensada para ser lo más económica posible, por lo tanto los diseñadores tuvieron que tomar ciertas decisiones para mantener esa línea, y esas decisiones se vieron reflejadas en su diseño final, lamentablemente no está diseñada para tener muchas interfaces que le permitan comunicarse con variados tipos de hardware. Si se da el caso que se requiera que la Raspberry trabaje con otro tipo de hardware para el cual no está diseñada, entonces lo mejor es buscar alguna otra solución.

Para trabajar con video, la Raspberry utiliza cualquier cámara con conector CSI-2 (el módulo de video es un Omnivision 5647), capaz de tomar fotos de hasta 5MP (2592x1944) en formato JPEG, PNG, GIF, y BMP, así como formatos no comprimidos como YUV o RGB o grabar videos (1080p, 1920x1080 a 30fps) y utiliza 250ma de corriente.

Este dispositivo es alimentado por un Micro USB a 5v y no permite PoE (Power over Ethernet). Exactamente ¿cuanta corriente necesita?, depende de los dispositivos que tenga conectados. Típicamente, la modelo B utiliza entre 700mA y 1000mA y la Modelo A cerca de 500mA. La máxima corriente que la Raspberry puede utilizar es 1A. Los GPIO, en total utilizan cerca de 50mA, el puerto HDMI usa cerca de 50mA, la cámara utiliza 250mA y los teclados pueden consumir desde 100mA hasta 1A. Para utilizar la Raspberry PI con baterías se debe tener cierto cuidado, por ejemplo, 4 pilas AA recargables generan 4.8volt, lo cual está justo en el límite de lo necesario para que la Raspberry funcione bien, sin embargo, el sistema se volverá rápidamente inestable puesto que la batería perderá su carga completa. Por otra parte, 4 pilas AA alcalinas (no recargables) resultarán en 6v lo cual está por encima de lo aceptado o necesario por la Raspberry lo que podría causar inestabilidad en el sistema o incluso destruir componentes de la Raspberry. En la misma línea de coste energético, la Raspberry PI también ha sido foco de investigación en la generación de clusters computacionales de baja huella de carbón y altas prestaciones [88].

En cuanto al soporte de software, la Raspberry PI funciona principalmente con distribuciones Linux, a pesar de que podría eventualmente ejecutar cualquier sistema operativo basado en x86, según la propia organización, no es recomendable utilizar Windows puesto que no satisface los requerimientos

mínimos para ejecutarlo. La distribución de Linux oficial para la Raspberry, es una basada en Debian llamada Raspbian, dicha distro es optimizada por los ingenieros de raspberrypi.org. Respecto de los lenguajes de programación a utilizar sobre la Raspberry PI, se recomienda utilizar Python, pero en general, cualquier lenguaje que pueda ser compilado para ARMv6 puede ser utilizado, ya sea Python, C, C++, Java, Scratch, Ruby y Wolfram, todos vienen instalados por defecto en Raspbian. El software es instalado en una tarjeta SD (o micro SD en los modelos +), se recomienda como mínimo contar con una tarjeta de 4GB clase 4 aunque puede utilizar una de mayor capacidad, aparentemente con más de 32GB también funcionaría, pero todo está bajo testing por el momento, en este estudio particular, se utilizó una tarjeta SD de 8GB clase 4. Respecto de la conectividad, la Raspberry Modelo B posee una conexión cableada Ethernet de 10/100mbps, no posee Wi-Fi (sin embargo es posible agregar un módulo Wi-Fi USB para otorgar conectividad inalámbrica), lamentablemente no está en los planes agregar soporte puesto que el SoC no soporta Wi-Fi, además esto incrementaría los costes lo cual está fuera del concepto Raspberry Pi, no posee una red GigaEthernet puesto que el canal de comunicación de la red usa el bus de USB 2.0 que no otorga un ancho de banda mayor.

En función de las características descritas y apoyados en algunas conclusiones obtenidas desde [89] quienes indican que la Raspberry PI modelo B puede ser utilizada con una carga relativamente liviana para mantenerla estable, y dado lo similar en tamaños y pesos así como también en potencia de cálculo, consumo energético (donde el modelo A/A+ es mejor pero con menos prestaciones), el modelo B presenta la mejor relación precio-tamaño-peso-calidad-funcionalidad (cabe destacar que el modelo B+ apareció después de realizar las compras para el proyecto). Además la mayor ventaja de la Raspberry PI y que la diferencia de Arduino por ejemplo, es la capacidad de ejecutar un sistema operativo, como Linux, lo que permite la instalación de software complejo para realizar tareas mucho más complejas que las que se podrían realizar con Arduino con mucho menos trabajo.

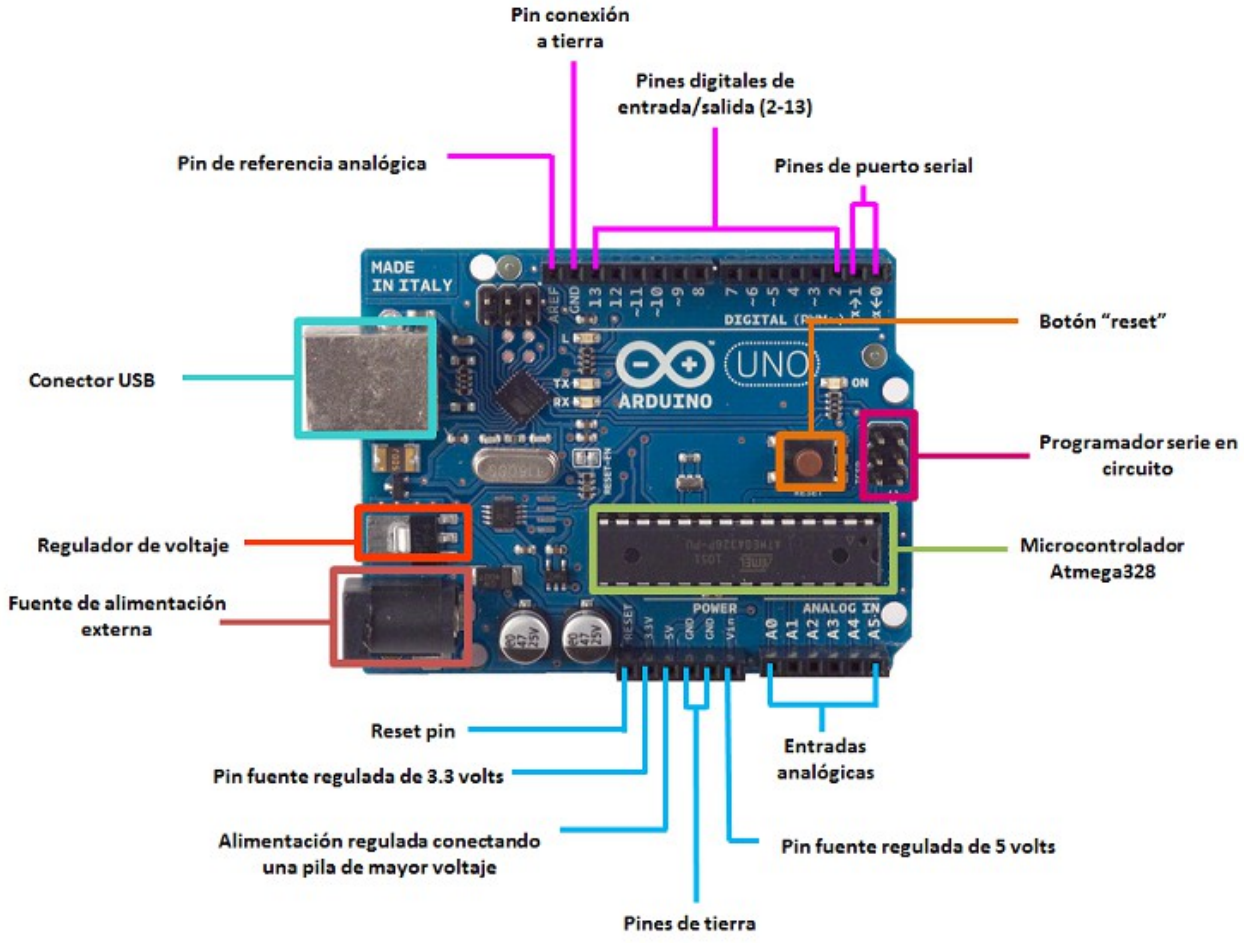
# Arduino Uno R3

Debido a que parte del sistema de control y las LCU serán una mezcla entre Raspberry PI y Arduinos, entonces a continuación se mostrarán algunas características principales del Arduino Uno R3 que es el hardware que se utilizó para controlar algunos dispositivos de hardware de muy bajo nivel.

## Overview

El Arduino Uno R3 es un microcontrolador basado en el Atmega328. Posee 14 pines de entrada/salida digital (de los cuales 6 pueden ser usados como salidas PWM (Pulse-Width Modulation)), posee 6 entradas análogas, un resonador cerámico de 16mhz, una conexión USB, un conector de energía, un header ICSP (In Circuit Serial Programming) y un botón de reset.

El arduino uno se diferencia del resto de las tarjetas de microcontroladores porque no hace uso de un chip FTDI USB a serial, en vez de eso, posee el Atmega16U2 programado como convertidor USB-serial. En la 37 se puede apreciar la placa Arduino Uno en su vista frontal.



## Energía

El arduino, uno puede ser alimentado usando la conexión USB o con una fuente de alimentación externa. La fuente de alimentación es elegida automáticamente por el arduino. La alimentación externa (no USB) puede venir de un adaptador AC a DC o una batería. El adaptador puede ser conectado a una entrada de 2.1mm. Si la energía viene de una batería, esta puede ser conectada directamente a los pines Gnd y Vin del conector de poder.

El arduino Uno R3 puede operar con una fuente de alimentación externa de entre 6 a 20 volts, sin embargo si se le entrega al arduino menos de 7 volts entonces las salidas tendrán menos de 5 volts y la tarjeta puede quedar inestable. Si se le entrega mas de 12 volts, el regulador de voltaje se podría sobrecalentar y dañar la placa, por lo tanto el rango recomendado es entre 7 y 12 volts.

El microcontrolador ATmega328 tiene 32kb de memoria, de los cuales 0.5kb son usados por el bootloader. Además posee 2kb de SRAM y 1kb de EEPROM.

Todos los 14 pines digitales en el Arduino Uno R3, pueden ser usados como entrada y como salidas. Cada uno de ellos opera a 5 Volts. Pueden proveer o enviar un máximo de 40 mA y poseen una resistencia variable (desconectada por defecto) de entre 20-50 kOhms. Además algunos pines poseen funciones especializadas:

- **Serial: 0(RX) y 1 (TX)**. Usados para recibir (RX) y transmitir (TX) data serial TTL. Estos pines esta conectados a los correspondientes pines en el ATmega8U2 USB-a-TTL chip serial.
- **Interrupciones Externas: 2 y 3**. Estos pines pueden ser configurados para lanzar una interrupción cuando se ve lee valor bajo, cuando un valor cambia, cuando un pin incrementa o disminuye su valor en voltaje.
- **PWM: 3,5,6,9,10 y 11**. Proveen una señal PWM de 8 bits.
- **SPI: 10 (SS, Slave Select), 11 (MOSI, Master Out Slave In), 12 (MISO, Master In Slave Out) y 13 (SCK, Serial Clock).** Estos pines soportan comunicación SPI (Serial Peripheral Interface).
- **LED: 13**. Posee un led incorporado que esta conectado el pin 13. Cuando el pin esta en Alto, el led se enciende, y cuando el pin esta en bajo, el led se apaga.

El Arduino Uno R3 posee 6 entradas análogas, etiquetadas como A0 hasta A5, cada una de ellas posee una resolución de 10 bits. Por defecto, miden desde tierra hasta 5 Volts, sin embargo es posible cambiar el límite superior e inferior usando el pin AREF. Además algunos pines análogos también poseen funciones especiales:

- **TWI: A4 o pin SDA y A5 o pin SDL**. Soportan comunicación TWI (Two Wire Interface, usada para $I^2C$)

## Comunicación:

El Arduino Uno R3 posee varias facilidades para comunicarse con un computador, con otro Arduino o con otros microcontroladores. El ATmega328 provee una comunicación serial UART TTL (5V), la que esta disponible en los pines 0 (RX) y 1 (TX). El Arduino permite comunicación vía USB con el computador, estos aparecen como puertos COM en Windows o como /dev/ttyACMX en GNU/Linux y se utilizan los drivers estándar de comunicación USB.

## Programación del Arduino:

El Arduino puede ser programado con el software de Arduino. El microprocesador ATmega328 viene preprogramado con un bootloader que permite cargar código nuevo al Arduino sin usar un programador externo físico. Se comunica usando el protocolo STK500.

Además, podría hacerse un bypass del bootloader que viene por defecto y programar el microcontrolador directamente usando el ICSP usando Arduino ISP o algo similar. Además es posible realizar un cambio completo de firmware.

## Resúmen:

| | |
|---|---|
| Microcontrolador | ATmega328 |
| Voltaje de Operación | 5B |
| Voltaje de entrada (recomendado) | 7 – 12 V |
| Voltaje de entrada (limites) | 6 – 20 V |
| Pines Digitales I/O | 14 (de los cuales 6 poseen salida PWM) |
| Pines de entrada Análoga | 6 |
| Corriente DC por pin I/O | 40ma |
| Corriente DC por pin 3.3V | 50ma |
| Memoria Flash | 32KB (ATmega328) de los cuales 0.5kb son usados por el bootloader |
| SRAM | 2 kb (ATmega328) |
| EEPROM | 1 kb (ATmega328) |
| Velocidad del reloj | 16 mhz |
| Largo | 68.6 mm |
| Ancho | 53.4 mm |
| Peso | 25g |

# CAPÍTULO IV

## Planteamiento del Problema

---

*En este cuarto capítulo, se realiza una descripción del problema que se desea resolver con el desarrollo de la presente investigación y posteriormente se definen los principales objetivos que deben ser logrados como resultado del trabajo desarrollado.*

# Planteamiento del Problema

El observatorio a ser construido por la Universidad de Antofagasta, con el proyecto Quimal adjudicado (QUIMAL130004) será ubicado en las cercanías de la ex estación Yungay, a 90km de Antofagasta. Al igual que muchos otros observatorios, se requiere que sus instalaciones se encuetren alejadas de los centros urbanos para evitar la contaminación lumínica. La pregunta esencial es, ¿Cómo se le dará uso a este observatorio? ¿Los estudiantes, científicos e investigadores tendrán que viajar los 90km para hacer uso de las instalaciones? ¿Trabajarán toda la noche allá?. Por otra parte, es un hecho que la teleoperación y observación remota de un observatorio puede disminuir los costos de operación del mismo [89]. En respuesta a esas inquietudes, existe la posibilidad de automatizar el observatorio, de tal manera que no requiera de un operador humano para funcionar.

A lo largo de los años, la automatización ha tenido un rol importante en la industria, permitiendo automatizar procesos y hacerlos mas eficientes, menos costosos y mas robustos. Por lo mismo, no es extraño pensar que en el ámbito de la ciencia, suceda algo similar, y en particular en la astronomía. Donde el control de telescopios debe ser tan preciso como para poder guiar una gran y pesada estructura con una cámara tomando una imagen con una exposición de 1 hora y mantener siempre la misma imagen centrada.

Una vez planteada la idea de automatizar el observatorio, nace la siguiente pregunta, ¿Cómo automatizar el observatorio?. Si bien es cierto se dijo anteriormente que el observatorio estaría ubicado en un lugar alejado de cualquier urbe, esto implica que al ser ubicado lejos, para operar existen dos opciones, dejar a un operador las noches que se requiera observar (para revisar que no existan percances con el domo, verificar que el domo se cerro como corresponde, verificar que la montura no esta atascada, revisar que el clima sea apto para observar, por dar algunos ejemplos) o automatizar el observatorio para que este pueda trabajar sin necesidad de tener operador humano.

Si bien es cierto, existen software para astronomía, la mayoría requiere de un alto conocimiento sobre dicho software para poder adaptarlo a los requerimientos de un observatorio en particular o bien construir el observatorio pensando en que funcione con un determinado software o framework. Sin embargo, cada observatorio tiene sus propios requerimientos. Muchas veces los software de astronomía deben ser altamente adaptados para satisfacer los requerimientos científicos y lo que es peor aun, tener mantención constante del software (para dar soporte es imperante conocer lo suficientemente bien el software que se está utilizando).

La presente tesis tiene como objetivo, demostrar que es posible utilizar en Astronomía un framework de robótica llamado ROS, ampliamente utilizado por una comunidad bastante activa de desarrolladores, la ventaja que esto presenta, es que al ser un software utilizado por una gran cantidad de investigadores para el desarrollo de la robótica, es muy robusto y existe mucha documentación al respecto de variados autores. Por otra parte, al ser ampliamente utilizado en robótica, muchos de los problemas de computación distribuida (al tratar un robot como un computador) ya están resueltos, y muchos de los

software de astronomía disponibles se basan en computación distribuida, por lo tanto, ROS cumple con los requisitos básicos para el control de un observatorio.

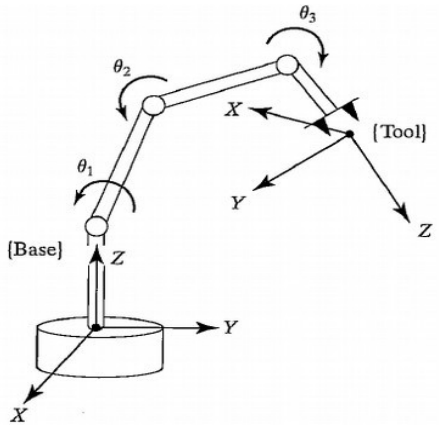
Al ser una herramienta libre y gratuita, no se debe invertir recursos en licenciamiento, lo que es un factor a considerar bastante importante sobre todo para observatorios de bajo presupuesto. Otra de las ventajas que presenta el software libre, es la posibilidad de arreglar o modificar el software según los requerimientos así como también, permite visualizar el código del framework que se está utilizando para realizar el control y hacer mejoras que luego se podrían transformar en una contribución al mismo framework.

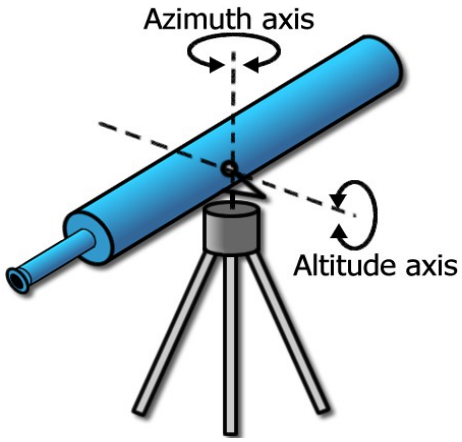
Como caso de estudio, la adaptación de este software se implementó en el observatorio construido por la Universidad de Antofagasta con fondos QUIMAL. Cabe destacar que al buscar en diferentes sitios bibliográficos acerca de implementaciones de ROS en Astronomía, no fue encontrado ningún trabajo en este ámbito, además la lista de paquetes disponibles en ROS no contempla en ningún caso, el control de algo relacionado con astronomía, por lo tanto sería un enfoque nuevo para este framework.

# Hipótesis

## Preámbulo:

Los manipuladores robóticos (ver 38) son un conjunto de piezas unidas por unas “articulaciones” artificiales y poseen ciertas características de movimiento y cinemática muy similares a una montura para telescopio (independiente del tipo de telescopio) del tipo altazimutal (ver 39).




Con lo cual uno podría pensar en que un telescopio de tipo altazimutal, es análogo a un brazo robótico de **2 grados de libertad**, sin embargo, al visualizar esta similitud, es posible entonces pensar en que un manipulador pudiese ser del tipo polar, y entonces uno podría pensar en que también un telescopio con montura ecuatorial, podría ser vista como un manipulador robótico en este caso de 3 grados de libertad, por lo tanto es posible pensar que un software de robótica, con lo drivers e información correcta podría ser utilizado para controlar un telescopio.

Sin embargo independiente de las características físicas de un telescopio, además podemos encontrar diversas similitudes en el software de control utilizado, por un lado, como se discutió previamente, todos los software de control en astronomía están diseñados para realizar computación distribuida, esto es utilizar diversos computadores como un todo, y entienden a un telescopio como una unidad computacional, en otras palabras como un computador, y es este computador quien interactúa con el hardware (telescopio y/o observatorio), haciendo las veces de front-end. Por otro lado, el framework de robótica particular propuesto para realizar este proyecto, es un software de computación distribuida que entiende por nodo o robot, a un computador que esta dotado o que interactúa con un hardware que a su vez interactúa con el medio físico, con lo cuál, las similitudes en cuanto a funcionalidad son notables.

De acuerdo a los último reportes de la ESO [1], la organización está tratando de migrar hacia dispositivos industriales que le permitan un mejor estado del arte en hardware, dado que las empresas que fabrican algunos de los dispositivos específicos para astronomía dejan de producirlos. Por lo mismo, adaptar un software de robótica, que en si ya cuenta con varios algoritmos de control y drivers para dispositivos industriales, podría ser una buena opción siguiendo la linea de desarrollo propuesta por ESO.

Dados los hechos remarcados anteriormente, la hipótesis propuesta para esta tesis es:

**Es posible, dada las analogías entre un software de control para astronomía y uno para la robótica, implementar el control automático y la teleoperación de un Observatorio, utilizando el software para robótica llamado ROS.**

El presente proyecto tiene como objetivo analizar como la tecnología orientada a la robótica podría ser utilizada en la astronomía, como una forma de apoyar el desarrollo de esta ciencia. Como problema anexo, estos centros de observación están ubicados en locaciones alejadas de los centros urbanos, por lo que requieren ser teleoperados, por lo tanto en esta tesis se busca investigar sobre las opciones de teleoperación para observatorios y luego analizar si esto es factible utilizando ROS.

# Objetivo

## General

Aplicar ROS para el control y la automatización de Telescopios y Observatorios, y desarrollar los drivers necesarios para la interacción del telescopio con el framework.

## Específicos:

1. Analizar software de astronomía.
2. Estudiar ROS.
3. Analizar compatibilidad de hardware con Linux y drivers asociados.
4. Diseñar drivers e integración con ROS.
5. Desarrollar Sistema SCADA.
6. Realizar la implementación en el observatorio.

# CAPÍTULO V

## Planteamiento de la solución al Problema

---

*En este capítulo, se realiza una detallada descripción sobre la solución planteada a los problemas encontrados. Además, se describe un caso de estudio, utilizando esta solución.*

# Planteamiento de la Solución al Problema

El sistema desarrollado posee el mismo principio utilizado para el control de un robot móvil. En particular, se usará como ejemplo uno de los robots mas populares basados en ROS, el Turtlebot II. El robot posee un computador a bordo que se encarga de la comunicación con el hardware, le indica a los motores cuando detenerse, cuando avanzar, se encarga de leer los sensores y de realizar acciones en función de la información leída desde los sensores. Básicamente, el robot posee capacidad computacional para tomar decisiones locales al poseer un computador a bordo. Esto lo convierte en un nodo computacional. Luego, si se establece la analogía, en la astronomía existen las LCU (Local Control Unit) encargadas del control de unidades o equipamientos puntuales. Las LCU, para lograr integrarse con las operaciones del observatorio deben poseer entendimiento mínimo del software utilizado para el control del observatorio, lo que convierte a las LCU en nodos computacionales con capacidad para entender y operar ciertos componentes o equipos. Visto de esa forma, la primera analogía encontrada es que un robot y un observatorio poseen esquemas similares de operación de hardware, ambos intentan conectar el hardware con un nodo computacional que le permita realizar operaciones de alto nivel sobre dicho hardware.

El robot es capaz de operar de forma reactiva, puesto que posee poder de computo, pero este es limitado, si bien es cierto, estos robots pueden comunicarse entre ellos, el poder de computo que poseen no es lo suficientemente robusto como para poder realizar una planificación grupal, dirigir y diseñar el plan de trabajo en equipo para uno o varios robots. Lo cual directamente trunca la autonomía de un robot, por lo mismo, el robot Turtlebot es capaz de comunicarse con un ROS Master a través de TCP/IP via WiFi, por ejemplo, dicho ROS Master está ubicado en otro equipo, de mucho más poder de cálculo, de esta forma, ahora este robot, que es un nodo computacional, ha entrado a un sistema de computación distribuida. En el caso de las LCU sucede algo similar, es demasiado costoso poner un equipo de alta potencia cerca de equipos móviles o que pudieran dañarse, por otra parte estos equipos de alto costo deberían comunicarse con otros equipos y así el observatorio podría poblarse de equipos costosos. Para resolver este problema, las LCU se conectan con su respectivo servidor, si la LCU es parte de un instrumento o una cámara, ésta debería comunicarse directamente con su IWS (Instrument WorkStation), si es una LCU relacionada con el movimiento del domo o el telescopio, ésta debería comunicarse directamente con el TCS. De esta forma, el concepto de LCU ahora está inmerso en un sistema de computación distribuida donde el sistema de control general, se comunica con estos dispositivos para poder darles instrucciones en función de obtener un objetivo particular.

Por último, un usuario podría querer desplazar un conjunto de robots desde una posición en particular hasta un punto X determinado por el usuario, luego el usuario se comunica con el sistema que controla a los robots y le indica a este sistema la necesidad de que ciertos robots se muevan hasta cierto punto y a cierta velocidad. El usuario en ningún momento debe preocuparse por mover cada uno de los robots, ni siquiera debe preocuparse por qué tipo de robots son y en qué lugar se encuentra cada uno, sólo sabe que desea una cierta cantidad de robots en un punto determinado. En el caso de la astronomía sucede

algo similar, el astrónomo sabe que quiere ver una estrella, que se encuentra en tal posición del cielo y debe ser seguida a una cierta velocidad, el astrónomo no está preocupado de mover el domo, la montura, tomar la imagen y medir la temperatura ambiental (por dar algunos ejemplos), el astrónomo sólo le dice al sistema lo que quiere hacer y este sistema debe ser capaz de entender sus instrucciones y determinar todos los movimientos necesarios en todos los equipos involucrados en la observación para lograr el objetivo. Por lo tanto, en ambos casos, existe un sistema que es capaz de entender las instrucciones de un operador humano y transformar estas ordenes en pequeñas instrucciones para cada uno de los subsistemas.

Por lo tanto, usando el mismo principio de control de un robot móvil, es posible realizar el control de un Observatorio y por ende ROS es perfectamente aplicable para el control de Observatorio.

## Arquitectura de software

Considerando que el software desarrollado utilizará como framework ROS, entonces éste debe considerar las normas de diseño que propone ROS. De esta forma, para el proyecto se creó un workspace llamado "*quimal*" y dentro se crearon 3 paquetes, cada uno con una funcionalidad particular (siguiendo las sugerencias de desarrollo de ROS), en la 40 se muestra la arquitectura del proyecto desarrollado:

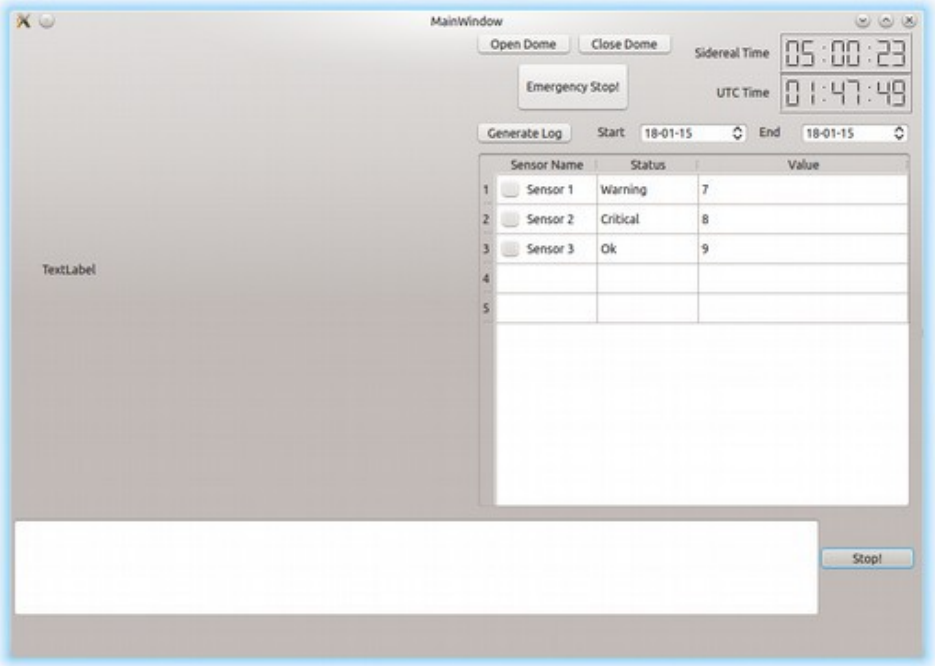
```
$ tree -L 1 quimal/
quimal/
├── build
├── devel
├── README
└── src
```

*Ilustración 40: Estructura de Workspace*

Todos los workspaces de ROS tienen la misma estructura, en el directorio *src* se almacenan los paquetes desarrollados, estos tienen cada uno, una función específica y están auto-contenidos para poder ser usados de forma independiente en otros proyectos.


*Ilustración 41: Estructura de paquetes*

En la 41 se muestran los tres paquetes principales: dome, sensor y telescope. Cada uno de estos paquetes funcionan de forma independiente y se encargan de una tarea puntual. De esta forma, si algún otro observatorio requiere la estructura de control de un domo, puede hacerlo sin necesariamente trabajar con el paquete de sensores o de telescopio. Además, podría usar esa estructura para otros fines ligados o no a la astronomía. Por otra parte, como arquitectura de software, se utilizó el Modelo Vista Controlador (MVC) lo que permite modularizar aún más el diseño del software y así quedan completamente aisladas las funciones principales del software, de los componentes gráficos. Por ejemplo, para el control del domo, se crearon 2 interfaces, una gráfica utilizando PyQt (ver 42) y una interfaz sencilla de comandos, ambas interfaces hacen uso exactamente de las mismas funciones de control, y si se quisiera luego desarrollar otro tipo de interfaces, bastaría sólo con usar las funciones que quedaron disponibles para el uso, provistas por el Controlador. Por otra parte, la integración de nuevas funcionalidades, es una tarea más sencilla puesto que cada nueva aplicación de control se desarrolla en el sub-paquete de control, y luego se implementan las funciones publicas que hacen accesible esa nueva función. Finalmente todo queda ordenado y comprensible para cualquier otro desarrollador. El paquete domo se puede apreciar en la 43.




```
├── CMakeLists.txt
├── msg
│   └── roof.msg
├── package.xml
├── src
│   ├── controller
│   │   ├── client
│   │   │   └── dome_controller_client.py
│   │   ├── controller.py
│   │   └── server
│   │       └── dome_controller.py
│   └── view
│       ├── console
│       │   └── dome_console.py
│       └── graphical
│           ├── dome_controller_gui.py
│           ├── dome_controller_UI.py
│           ├── sensor_table_model.py
│           └── tcs_gui.ui
└── srv
    ├── close_dome.srv
    └── open_dome.srv
```

*Ilustración 43: Estructura del paquete dome*

La arquitectura de software, se rige con las normas de desarrollo propuestas por ROS, por la misma razón, todo pareciera estar desfragmentado, sin embargo, esta misma filosofía de desarrollo ha permitido reutilizar varios componentes existentes en ROS para el desarrollo del observatorio, un caso puntual es el paquete image_transport, diseñado originalmente para enviar imágenes desde un robot y hacer ciertas operaciones con estas. Éste ha sido utilizado dentro de un sensor para enviar imágenes al TCS, esto ahorró mucho tiempo de desarrollo, lo cuál marca la importancia de escoger correctamente el framework.

Uno de los grandes problemas detectados con ROS, fue la forma en que utiliza los puertos de comunicación. Para trabajar remotamente con instancias de ROS, se generan comunicaciones sobre diferentes puertos, cada puerto está asociado a un nodo lógico de ROS (programas que escuchan por un puerto), por lo tanto, para trabajar con instancias de ROS distribuidas a través de internet, cada nodo computacional de ROS debería tener todos sus puertos expuestos a internet para que cada uno pueda acceder al servicio de ROS que necesita.

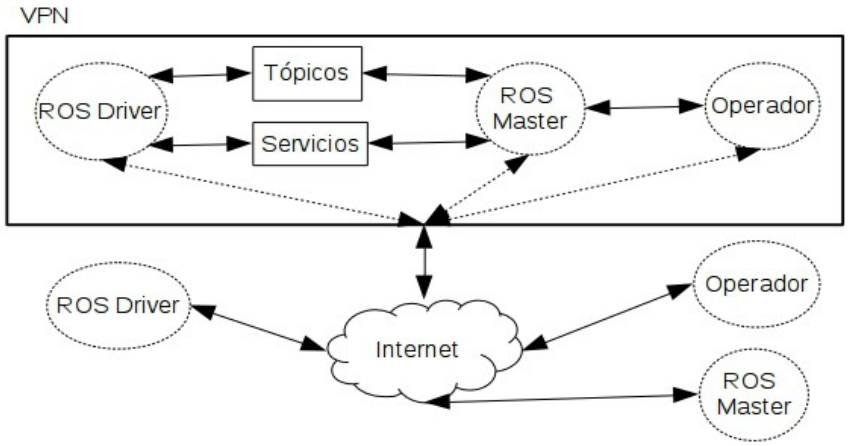
Asumiendo el caso del Robot Turtlebot II explicado anteriormente, se podría pensar que exista un servidor que posea el ROS Master ubicado en el sitio A, luego se podría tener al robot en el sitio B y por último a un operador en el sitio C. Para que los nodos distribuidos geográficamente puedan comunicarse a través de internet usando ROS estos deberían ser nodos públicos con los rangos de puertos definidos libres para ROS lo que implica que cualquiera podría ver y peor aún modificar estados, condiciones, parámetros y/o mover el robot sin permiso. Dado que ROS es completamente abierto y sin seguridad, entonces cualquier nodo se puede subscribir a un tópico y publicar o leer información.

Para resolver el problema antes mencionado, se pueden presentar dos soluciones, la primera es que si bien es cierto cualquier nodo podría publicar un mensaje en un tópico, si este mensaje no posee un id o un hash que lo identifique, entonces el mensaje no es leído o es descartado por el nodo que lea el tópico, la misma situación para los servicios, uno podría pensar en pedir un hash o un id que certifique que el nodo que está realizando la petición sea el que corresponde y así descartar peticiones de otros nodos. Sin embargo, estas soluciones son susceptibles a ataques del tipo DDOS, simplemente publicando miles de mensajes en un tópico lo que implicaría que el subscriptor tendría que leer miles de mensajes basura antes de leer un mensaje útil y peor aún podría colapsar el ROS Master dejando incomunicado a todo el sistema. Por otra parte, los servicios podrían dejar de responder al estar ocupados rechazando nodos que tratan de comunicarse.

Como se explicó anteriormente dejar los nodos expuestos podría causar muchos problemas a un sistema basado en ROS, por lo tanto existe una forma mucho mas sencilla de aislar el sistema sin quitar la conectividad a través de la red, una VPN, con esta sencilla solución, solo los nodos que estén permitidos dentro de la VPN y que pertenezcan a la misma red podrían hacer uso del sistema tal y como si estuvieran de forma local trabajando, de esta forma, todos los nodos que están con sus puertos

libres y accesibles dentro de la red local, pueden funcionar de forma distribuida y seguiría siendo seguro mientras ningún nodo extra logre entrar a la VPN. Cabe destacar que si un nodo comienza a publicar de forma indebida en un tópico, para hacerlo primero tendría que estar en la VPN lo que significaría que fue considerado en el diseño para estar dentro de la VPN, por lo tanto es un error de diseño mas que una vulnerabilidad de seguridad.

Si aplicamos este concepto al ejemplo del sistema distribuido para el Turtlebot II, los 3 nodos distribuidos en A, B y C, simplemente deberían conectarse a la VPN una vez que tengan acceso a internet, sin importar si están detrás de un firewal, un proxy o uno o varios nat. Una vez lograda la conexión hacia la VPN, ambos estarán inmersos en la misma red local y podrán comunicarse tal y como si estuvieran en la misma red física. De esta forma, todo nodo que entra a la VPN es capaz de trabajar en el entorno de ROS, y luego si el robot y el ROS Master están operativos; para poder operarlos desde cualquier lugar, sólo se tiene que conectar el computador que posee el programa para teleoperar el robot a la VPN, realizar las configuraciones que sean necesarias (dependiendo de si la VPN tiene IPs fijas o dinámicas) y operar el robot. En la 44 se explica gráficamente el funcionamiento del sistema usando ROS y VPN.




En el caso de un observatorio, la situación es idéntica, pero con mayor justificación, el observatorio tiene que estar alejado de la ciudad para evitar la contaminación lumínica, por lo tanto al estar ubicado en un área remota, la mejor forma de operar el observatorio es teloperarlo (puesto que los costos de mantener personal en el sitio son altos). Más aún, si se quisiera permitir a astrónomos de todo Chile realizar observaciones, entonces, debería existir alguna forma de operar el observatorio de forma segura desde sitios alejados del observatorio. En esta línea, para permitir a un observador operar el observatorio, simplemente se le debe permitir el acceso a la VPN y proveer al usuario, el software que

permite realizar el control, de esta forma, el software sería un cliente remoto del observatorio y luego se tendría una situación similar a la del ejemplo del Robot, por un lado está el ROS Master funcionando en un equipo; hay varias LCU distribuidas por el observatorio controlando diferentes componentes, estos se comunican con el ROS Master y a su vez, la aplicación que controla el observatorio, que está en otro equipo diferente, en una ubicación diferente se comunica con el ROS Master, exactamente la misma situación planteada para el ejemplo del Robot pero aplicado a un observatorio.

Cada nodo de ROS se registra en el ROS Master utilizando un nombre que debe ser único, si se diera el caso que múltiples usuarios intentan hacer uso del mismo recurso, como por ejemplo la aplicación de control, primero se debe terminar la conexión de la aplicación activa, de otra forma ROS no permite generar dos instancias del mismo nodo con el mismo nombre y por consecuencia termina el nodo que se encontraba activo dejando sólo el ultimo nodo conectado, esto significa que quien estaba operando el observatorio pierde el control del sistema porque alguien más se conectó. Por otra parte, ROS es flexible y por tanto el nombre que utilizará el nodo puede ser cambiado, esto significa que se podrían tener múltiples interfaces de control para el mismo telescopio. Las ventajas de esto es que pueden haber diferentes operadores revisando el estado del TCS, por ejemplo operadores locales y remotos, la desventaja, si no hay coordinación se podría dar el caso de que el telescopio reciba dos ordenes simultáneamente. Para abordar esta situación, las ordenes deben ser controladas a través de servicios, lo que significa que una vez terminada la ejecución de la primera orden, se procederá a ejecutar la siguiente. Este comportamiento no dista mucho de otros observatorios como Paranal, donde se pueden tener dos o mas paneles de control para el TCS, ambos pueden enviar comandos al telescopio, por eso hay un fuerte trabajo de coordinación entre los equipos trabajando en el telescopio. En términos prácticos, las interfaces de control sólo envían ordenes, pero son los dispositivos o controladores quienes deciden qué ejecutar, cuándo y cómo en función del modo de operación que se les defina.

Actualmente la conectividad del observatorio está pensada utilizando una red móvil 3G, con un plan de datos, lo que presenta dos grandes restricciones, la primera, la cuota límite de tráfico y datos, lo que significa que los datos deben ser optimizados para no consumir todo el ancho de banda otorgado durante el mes. Por otra parte, aún cuando la señal es bastante buena y rápida, no existe una real certeza de cuán confiable es dicha red, por lo tanto siempre existe el riesgo latente de que la comunicación se detenga y por ende el observatorio quede aislado. Para resolver estos problemas, las políticas de seguridad del observatorio indican que si no hay conectividad con el rosmaster, simplemente el observatorio se cierra, no se puede correr el riesgo de que luego éste no sepa que hacer. La conectividad dentro del observatorio es mediante WiFi usando un router industrial Robustel 3PWR3000 con conectividad 3G.

Por el lado de la montura, se logró hacer una conexión serial con la montura Losmandy G11 y Celestron, en ambos casos se desarrolló un programa en C++ utilizando la librería Serial provista por ROS para establecer la comunicación con el dispositivo, cada montura posee un firmware que es capaz de entender comandos vía serial usando un protocolo que es diferente en cada montura. Cabe destacar que el hecho de que existiese una librería Serial en ROS, facilitó de gran manera el desarrollo de los

drivers para las monturas.

Finalmente para tener medición del clima, se utilizó la estación meteorológica Davis Vantage Pro 2, la cual posee una serie de sensores, entre ellos medición de radiación UV, velocidad de viento, humedad y temperatura por nombrar algunos. Para integrar esta estación meteorológica al sistema, se decidió instalar primero un software llamado Weewx [90]. Este software se comunica periódicamente con la estación meteorológica, obtiene los datos y crea una página web con los resultados pero además genera una base de datos con toda la información meteorológica, por lo tanto lo que hace el ROS driver desarrollado, es conectarse con la base de datos generada por Weewx y obtiene los datos desde ese lugar. De esta forma, se aprovechan las funcionalidades que otorga Weewx, pero además se logra la integración de la estación meteorológica al sistema sin involucrar tanto tiempo de desarrollo.

En la tabla 1, se muestra un cuadro resumen que posee el listado de los componentes que tuvieron que ser integrados a ROS y el estado de integración de ellos.

| **Equipo** | **Modelo** | **Integración** | **Comentarios** |
|---|---|---|---|
| Motores | Nice ROBUS 600 Nice RUN 1500 | Driver desarrollado en python utilizando GPIO. Integración exitosa. | Para lograr la integración de este componente fue necesario el desarrollo de driver electrónico, puesto que la salida de 3.3volt de la Raspberry PI no proveía la potencia suficiente para activar los controles de los motores. |
| Estación meteorológica | Davis Vantage Pro 2 | Driver desarrollado en python, se realizan consultas a una base de datos generada por weewx. Integración exitosa. | Como apoyo a este driver, está el software Weewx, es un servicio de Linux que actualiza constantemente la información de la estación meteorológica y la guarda en una base de datos. De esta forma el driver de ROS solo debe consultar la misma base de datos. |
| Telescopios | Montura Los Mandy y Celestron | Se desarrolló un driver en C++ para comunicarse vía serial con la montura. Integración exitosa. | Para los telescopios, se desarrollaron dos controladores para dos monturas diferentes, sin embargo el envío de comandos es siempre en el mismo formato, el ROS driver se encarga de resolver como transformar la información recibida al hardware que corresponde |

Tabla 1: Resúmen de integración de equipos a ROS

# CAPÍTULO VI

## Discusión de resultados

---

*En este capítulo se realiza una discusión y análisis de los resultados obtenidos, eficiencia, eficacia y utilidad de la solución propuesta.*

# Discusión de Resultados

Este proyecto a permitido demostrar la aplicabilidad de ROS para el control de un observatorio de manera remota, con la consecuente reducción significativa de los costos de adquisición de software propietario, posibilitando además el uso de un framework que dispone de una extensa variedad de software reutilizable, producto de un conjunto de drivers que hacen posible la interconexión del observatorio a controlar. Para el caso particular de esta tesis, el "***Observatorio Ckoirama***" es el caso de estudio. La demostración de este hecho en particular, permite a futuro la integración de ROS para el desarrollo de Software de Astronomía. Por ejemplo, el software desarrollado para el control del observatorio Ckoirama puede ser a futuro reutilizado en un 58% en el desarrollo de software de control para Observatorios que posean telescopios con monturas Celestron o LosMandy.

Producto del estudio del estado del arte en el desarrollo de Software para el control de telescopios, se pudo descubrir que existe una gran cantidad de observatorios que poseen una implementación de software propia. Por lo tanto, al utilizar ROS se intenta promover un software ampliamente utilizado en Robótica como medio válido para astronomía y con ello concentrar los esfuerzos de la comunidad en un solo lugar en vez de diversificarla.

Los drivers desarrollados en nuestro caso de estudio, corresponden a software de Arduino que interactúa con el hardware y permite hacer el control. Estos sistemas son conectados a una Raspberry PI que se encarga de hacer de interfaz entre el hardware y el software. Para el driver de telescopio, se desarrollaron 2 drivers, uno para monturas Celestron y otro para LosMandy.

Para cuantificar el esfuerzo en horas hombre involucrados en el desarrollo del sistema de control del observatorio Ckoirama, se obtuvieron los siguientes datos estadísticos (Tabla 2):

17 archivos python, 7 archivos para C++ (entre archivos .cpp y .h), 15 archivos de configuración (definición de mensajes y servicios). El código desarrollado para el sistema de control del observatorio es de 3.016 líneas de código entre archivos de Python, C++ y archivos de configuración. ROS por su parte considerando el framework base y los utilitarios instalados disponibles en los repositorios posee 344 archivos Python sumando un total de 79.935 líneas de código, 379 archivos C++ sumando un total de 73.818 líneas de código y 203 archivos de configuración equivalentes a 1.980 líneas de código y comentarios. A estas estadísticas no se agregaron los archivos .xml, .yaml, Makefiles, cmakes y cualquier otro archivo de configuración que no sea parte de la configuración para construir el sistema.

Según los valores calculados, podemos decir que el sistema desarrollado corresponde al 1.8% del código funcional de la plataforma, el otro 98.1% corresponde a las herramientas funcionales de ROS. Esto significa que la Universidad se debe hacer cargo de mantener solo un 1.8% del sistema y el resto viene por cuenta de la comunidad de ROS. Esto no quita que se puedan realizar aportes a la plataforma, pero significa que la cantidad de código a mantener es muy inferior en relación a las funcionalidades que presenta.

Las 3.016 líneas de código desarrolladas, se distribuyen de la siguiente forma:

| Tipo | Líneas de Código | Cantidad de Archivos |
|---|---|---|
| Archivos de Python | 1450 | 17 |
| Archivos de C++ | 1512 | 7 |
| Archivos de Configuración | 54 | 15 |

*Tabla 2: Estadísticas de código desarrollado*

De las estadísticas anteriores, C++ es el lenguaje que nos presenta la mayor cantidad de reusabilidad del software, puesto que se desarrollaron las interfaces con las monturas de telescopio en dicho lenguaje, lo que significa que el mismo software desarrollado para esta montura, puede ser utilizado en monturas de otros observatorios. La cantidad de código reusable y que constituye una contribución real al framework es de 1.755 líneas, lo que corresponde a un 58% del total del desarrollo y contempla los archivos en C++ y Python mas los archivos de configuración (mensajes y servicios del telescopio).

La tablas 3 y 4 corresponden a un cuadro comparativo entre cantidad de código desarrollado para la plataforma y el reutilizado de ROS:

| Tipo | Líneas de Código | | Total | Porcentajes | |
|---|---|---|---|---|---|
| | ROS | Ckoirama | | ROS | Ckoirama |
| Archivos de Python | 79.935 | 1.450 | 81.385 | 98,21% | 1,78% |
| Archivos de C++ | 73.818 | 1.512 | 75.330 | 97,99% | 2,00% |
| Archivos de Lisp | 8.125 | 0 | 8.125 | 100% | 0% |
| Archivos de Configuración | 1.980 | 54 | 2.034 | 97,34% | 2,65% |
| **Total** | **163.858** | **3.016** | **166.874** | **98,19%** | **1,80%** |

*Tabla 3: Comparativa de líneas de código entre el sistema desarrollado y ROS*

Tabla comparativa de Cantidad de Archivos.

| Tipo | Cantidad de Archivos | | Total | Porcentajes | |
|---|---|---|---|---|---|
| | ROS | Ckoirama | | ROS | Ckoirama |
| Archivos de Python | 344 | 17 | 361 | 95,29% | 4,70% |
| Archivos de C++ | 379 | 7 | 386 | 98,18% | 1,81% |
| Archivos de Lisp | 57 | 0 | 57 | 100% | 0% |
| Archivos de Configuración | 203 | 15 | 218 | 93,11% | 6,88% |
| **Total** | **983** | **39** | **1.022** | **96,18%** | **3,81%** |

*Tabla 4: Comparativa de cantidad de archivos entre el sistema desarrollado y ROS*

Finalmente podemos concluir que ROS es un gran aporte para el desarrollo de la Astroningenier*ía*, pues brinda todas las interfaces que permiten el desarrollo de un sistema de control para un Observatorio. Otra ventaja encontrada es el nivel de abstracción de software logrado, el framework permite fácilmente dejar todo lo relacionado con un dispositivo en un solo paquete, lo que permite un mejor desacoplamiento de código y por ende mucho más mantenible gracias a sus modularidad. Si bien es cierto, el objetivo detrás de un framework es precisamente obtener una conclusión de este estilo. Los resultados de este trabajo indican que la utilización de ROS como framework es viable para el desarrollo astronómico, y basados en la literatura no hay razón técnica ni conceptual que no permita a ROS ser utilizado para la Astronomía

Segun Google Scholar, ROS [21] posee mas de 2000 citaciones mientras que el software más popular en Astronomia Amateur, ASCOM tiene no mas de 200. Basados en ese resultado, es mucho mas probable conseguir soporte y líneas de investigación para ROS que para algún otro software ligado a la Astronomía. Además especializar investigadores en un framework de Astronomía reduce su empleabilidad al ser éste, poco utilizado en el sector industrial, sin embargo especializar a un investigador en un framework de robótica amplia su horizonte al área de la automatización y robótica tanto en investigación como en la industria. Con las interfaces correctas y publicadas, ROS en astronomía podría llegar a ser tan funcional y operativo como otras plataformas de Astronomía, sólo depende del nivel de desarrollo que alcancen cada uno de los drivers apropiados.

Respecto de la teleoperación misma del observatorio, se obtienen muy buenos resultados en cuanto a envío y recepción de comandos, pero se debe hacer una clara mejora en el envío y recepción de imágenes. Con el fin de disminuir el tráfico relacionado a la operación, se decide enviar desde el observatorio imágenes de baja calidad con el objetivo de que sean lo mas fluidas posibles. Respecto de las interfaces gráficas, es preferible utilizar comandos por sobre interfaces gráficas para disminuir al máximo la latencia y el tráfico, sin embargo también es recomendable la generación de un sitio web que permita interactuar de forma gráfica pero con bajo tráfico de datos.

Actualmente, nos encontramos trabajando en la mejora de los sensores y la implementación de nuevas medidas de seguridad que garanticen una mejor operación, asi como también en la implementación de un sitio web que permita la teleoperación del observatorio desde cualquier tipo de cliente, sin necesidad de instalar componentes extra en la estación de trabajo del cliente.

La teleoperación del observatorio supone un mayor ahorro en cuanto a dinero y tiempo, puesto que la cantidad de viajes se reducen así como también los costos asociados, sin embargo introduce otros riesgos propios de la operación remota de cualquier dispositivo que deben ser considerados.

Los costos asociados a la investigación científica local se reducen, pero mejor aún, permitiría a científicos de otras partes del mundo poder enviar los objetivos para ser observados sin necesariamente estar físicamente en Chile. Finalmente se pueden utilizar las horas hábiles de países con usos horarios diferentes al de Chile, tal que el comienzo de la noche en Chile coincida con el inicio de la jornada laboral en otro país.

# CAPÍTULO VII

## Conclusiones y Trabajo Futuro

---

*En este capítulo se relatan las principales conclusiones obtenidas producto del desarrollo de una investigación de tipo aplicada. Es decir, la investigación y desarrollo de una solución de bajo costo para la teleoperación del Observatorio Ckoirama de la Universidad de Antofagasta. Este observatorio, se encuentra ubicado a mas de 90km de la ciudad de Antofagasta y la teleoperación de este observatorio, puede contribuir a la generación de ciencia a nivel regional, nacional y mundial. Posteriormente se enuncian diversos trabajos que podrían ser desarrollados en investigaciones futuras para mejorar aun más la utilización de ROS en astronomía.*

# Conclusiones

El objetivo de esta tesis de Magister, fue comprobar si ROS, un meta sistema operativo para robótica, podría ser utilizado para el control y teleoperación de un observatorio Astronómico. El estudio del estado del arte permitió descubrir que los observatorios funcionan como sistemas de computación distribuidos y tienden al uso de nuevos componentes para controlar la obsolescencia.

Para el desarrollo de la astronomía, es esencial contar con una excelente calidad de cielo, esto se logra en sectores remotos alejados de cualquier centro urbano para evitar la contaminación tanto lumínica como de radiofrecuencia, sin embargo, esto genera dificultades relacionadas con la operación y utilización de Observatorios Astronómicos. Para resolver estos problemas, estas edificaciones deben ser frecuentemente teleoperadas y robotizadas para minimizar los costos de operación.

Considerando la problemática presentada, se propuso utilizar ROS como framework para el desarrollo de un sistema de control y teleoperación del observatorio logrando implementar exitosamente un prototipo de teleoperacion para observatorios. De acuerdo a la literatura, la comunidad astronómica es reducida, distribuida y altamente especializada. Por lo tanto, utilizar un framework ampliamente utilizado en el campo de la robótica, brinda una solución de software económica y altamente soportada y diseñada para el control de robots. Actualmente la astronomía tiende a robotizar y automatizar los observatorios, por lo que utilizar un software de robótica para acelerar ese proceso, es el camino más indicado. La evolución de los observatorios puede perfectamente ir de la mano con la evolución de la robótica y la teleoperación.

Producto de la investigación realizada, se pudo determinar la estructura general de un observatorio astronómico así como sus requerimientos de funcionamiento. Todos los observatorios revisados, consideran sistemas de computación distribuidos, esto significa que cada dispositivo físico es entendido como un computador, y por tanto es posible controlar estos dispositivos usando un software. De forma general, lo que se utiliza, es una serie de aplicaciones cliente/servidor, aquellas que actúan como clientes son las que son utilizadas por los operadores para realizar las acciones en el sistema, mientras que las que funcionan como servidores son los drivers asociados a cada dispositivo a controlar, este comportamiento es la base de funcionamiento de ROS y también de la mayoría de los observatorios. Por otra parte, los sistemas de control están fuertemente ligados a las condiciones climáticas. En el sistema de control se consideran los sensores que realizan las mediciones ambientales, las que son determinantes para la observación, desde indicar si se puede abrir el domo, hasta medir cuanto es la humedad atmosférica, datos que son relevantes para el estudio de la imagen observada.

Sin embargo, la principal característica encontrada en este estudio, es que todos los observatorios tienden a la teleoperación, por la enorme reducción de costos que esto significa, y lo intentan desarrollando sus propias herramientas, siempre manteniendo el foco en la especialización con la que se caracteriza la astronomía. Por esta razón, que desde una mirada externa, se extrajeron los requerimientos principales de un sistema de control para observatorios y se contrastaron con ROS,

estos requerimientos fueron satisfechos en su totalidad, no existe alguna razón por la que ROS no pueda ser utilizado para astronomía. Con el desarrollo apropiado de una serie de drivers, se puede lograr un gran desarrollo y una alta utilización de esta plataforma en observatorios que no cuentan con los recursos para una solución comercial o simplemente, al igual que el caso de estudio, se tiene una estructura completamente nueva para la que no existe software comercial.

En esta investigación se generaron drivers que son altamente generales a nivel de software y solo varían en el control del dispositivo en particular, siendo fácilmente adaptables a cualquier tipo de observatorio. Además, se desarrollaron interfaces para telescopios que pueden ser utilizadas en conjunto con el resto del software o bien de forma aislada, contribuyendo con el estado del arte de control de monturas siguiendo la filosofía de ROS, ser un muy buen framework con una extensa variedad de herramientas.

Dentro de los dispositivos de control utilizados, se destaca las plataforma Arduino y Raspberry Pi, ambas presentan una gran versatilidad y pueden ser utilizadas en casi cualquier aplicación computacional, en el caso puntual de esta tesis, se utilizó la tarjeta Raspberry PI como LCU (Local control Unit) puesto que posee la habilidad de montar un sistema operativo Linux en el cual ejecutar los drivers de ROS para algún dispositivo, dependiendo del hardware a controlar, la tarjeta Raspberry Pi utilizaba Arduino en los controles análogos o bien los GPIO de la Raspberry Pi para salidas digitales.

La tarjeta Raspberry PI fue utilizada de forma inalámbrica a través de WiFi, un caso puntual de este hecho, es el controlador del domo, el que no posee conexión alguna con la base, por lo que se decidió utilizar una red inalámbrica en la comunicación. El resto de equipos que son estáticos, fueron conectados a un cable de red para en conjunto tener comunicación con el rosmaster. Finalmente se decidió instalar el rosmaster en el observatorio para mejorar la comunicación entre los nodos, y otorgar acceso via VPN sólo al software ROS Master, de tal manera que un operador externo ejecute los comandos de control directo en el ROS Master y éste a su vez se comunique con las LCU de forma local. El uso de VPN en las raspberry pi ralentizaba demasiado la comunicación y las acciones de control, por lo que se decidió rediseñar la solución para mejorar la velocidad de las acciones de control, por lo demás no es necesario que todas las LCU esten conectadas a la VPN si todos los equipos se encuentran en el mismo lugar, lo que importa de la teleoperación es la habilidad de poder controlar el dispositivo a distancia y eso se ha logrado sin entorpecer la operación misma del observatorio.

Las pruebas fueron exitosas, y se logró operar el observatorio a distancia utilizando la solución mencionada, sin embargo estas pruebas se realizaron considerando la energía en base a un generador, puesto que los trabajos de conexión a la Planta Solar aún no han concluido, es por tal razón que el sistema aún no está operativo y sólo está en fase de prueba, la operación sigue siendo in-situ hasta que se logre mantener un sistema de energía constante, en ese momento, entra en operación de forma remota el primer observatorio público chileno.

# Trabajo Futuro

Como trabajo futuro, se propone:

- Seguir mejorando el software de control del observatorio, si bien es cierto que la hipótesis fue comprobada, aún queda pendiente la publicación del software desarrollado como paquete de ROS bajo el nombre de “**ROS Astronomy**”.
- Diseñar e integrar nuevos drivers para controlar mayor cantidad de telescopios y agregar drivers para operar domos y sensores ya existentes en el mercado, la implementación actual consideró el diseño de todas las partes del control, por lo tanto no muchos de los componentes fueron comprados, pero para que este software sea replicable, puede que otros proyectos si requieran comprar equipamiento y por ende deberían estar desarrollados los drivers para el control de dichos equipos.
- En este proyecto, el enfoque estuvo en el control del observatorio, sin embargo no se ha mencionado el software de análisis de datos ni el de manejo de datos, ambos son perfectamente implementables bajo la filosofìa ROS, sería muy útil incorporar a este proyecto, algún sistema que permita recolectar los datos de la cámara y luego eficiente e inteligentemente distribuirlo al lugar que corresponda, reducirlos cuando sea necesario y determinar cuando enviar los datos fuera del observatorio.
- Tambien sería bastante útil tener weewx integrado a ROS, de esta forma se amplia el hardware soportado por ROS y además podría servir para monitorear el clima para observatorios.
- Finalmente se podría incluir un modelo 3D del observatorio en el simulador MORSE para luego leer las acciones de control del observatorio y visualizarlas gráficamente.

# Bibliografía


[1] G. Chiozzi, A. Bridger, K. Gillies, B. Goodrich, J. Johnson, K. McCann, G. Schumacher, and S. Wampler, “Enabling technologies and constraints for software sharing in large astronomy projects,” in *SPIE Astronomical Telescopes + Instrumentation*, 2008, p. 70190Y–70190Y–12.

[2] B. Sifón and C. Torres, “Más de US$ 3.000 Millones para poder mirar las estrellas desde los cielos mas puros de chile,” *Diario Financiero*, Santiago, p. 2, Mar-2013.

[3] “Very Large Telescope,” *www.eso.org*. [Online]. Available: http://www.eso.org/public/teles-instr/vlt/.

[4] “ALMA,” *www.eso.org*. [Online]. Available: http://www.eso.org/public/teles-instr/alma/.

[5] “The European Extremely Large Telescope | ESO,” *www.eso.org*. [Online]. Available: http://www.eso.org/public/teles-instr/e-elt/.

[6] “Presidente de Chile visita Paranal para anunciar transferencia de terrenos para el E-ELT | ESO Chile.” [Online]. Available: http://www.eso.org/public/chile/news/eso1345/. [Accessed: 25-Feb-2014].

[7] “LEY DE PRESUPUESTOS AÑO 2006.”

[8] “Ley de presupuestos año 2016,” 2016.

[9] CONICYT, “RESUMEN EJECUTIVO ELABORADO POR EL PANEL EVALUADOR E INFORME DE COMENTARIOS A LOS RESULTADOS DE LA EVALUACIÓN ELABORADO POR LA INSTITUCIÓN RESPONSABLE DEL PROGRAMA EVALUACIÓN PROGRAMAS GUBERNAMENTALES (EPG) FONDO NACIONAL DE DESARROLLO CIENTÍFICO Y TECNOL,” Santiago, 2013.

[10] www.astronomia.com, “Observatorio astronómico - Diccionario de astronomía.” [Online]. Available: http://www.astromia.com/glosario/observastronom.htm. [Accessed: 26-Jan-2014].

[11] J. Gates, W. T. S. Deich, A. Misch, and R. I. Kibrick, “Modern computer control for Lick Observatory telescopes,” in *SPIE Astronomical Telescopes + Instrumentation*, 2008, p. 70190C–70190C–12.

[12] R. J. Tobar, H. H. von Brand, M. A. Araya, and J. S. López, “An amateur telescope control system: toward a generic telescope control model,” in *Advanced Software and Control for Astronomy II. Edited by Bridger*, 2008, vol. 7019, p. 70192I–70192I–9.

[13] F. J. Aceituno, M. Abril, P. Luis, I. Bustamante, and M. Ubierna, “T35: a small automatic telescope for long-term observing campaigns,” 2010.

[14] K. Lanclos, W. T. S. Deich, R. I. Kibrick, S. L. Allen, and J. Gates, “Realizing software

longevity over a system's lifetime," in *SPIE Astronomical Telescopes + Instrumentation*, 2010, p. 77403E–77403E–12.

[15] J. M. Ibáñez Mengual, M. Fernández, J. F. Rodríguez Gómez, A. J. García Segura, and C. Storz, "The PANIC software system," in *SPIE Astronomical Telescopes + Instrumentation*, 2010, p. 77402E–77402E–12.

[16] W. Jian, J. Ge, Y. Xiaoqi, H. Kun, L. Feng, and R. Jian, "The Design of Observatory Control System of LAMOST," *Plasma Sci. Technol.*, vol. 8, no. 3, pp. 347–351, May 2006.

[17] Altavoz, "Crean innovador sistema de acceso remoto a telescopios | Diario Financiero Online."

[18] T. J. Ames and L. Case, "Distributed Framework for Dynamic Telescope and Instrument Control," in *Astronomical Telescopes and Instrumentation*, 2002, pp. 63–74.

[19] R. Cedazo, D. Lopez, F. M. Sanchez, and J. M. Sebastian, "The Montegancedo astronomical observatory: The first free remote observatory for learning astronomy," in *IEEE EDUCON 2010 Conference*, 2010, pp. 1367–1373.

[20] T.-C. Shen, R. Soto, J. Reveco, L. Vanzi, J. M. Fernández, P. Escarate, and V. Suc, "Development of telescope control system for the 50cm telescope of UC Observatory Santa Martina," in *SPIE Astronomical Telescopes + Instrumentation*, 2012, p. 84511T.

[21] M. Quigley, K. Conley, B. Gerkey, J. FAust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Mg, "ROS: an open-source Robot Operating System," *Icra*, vol. 3, no. Figure 1, p. 5, 2009.

[22] A. Hadravová and P. Hadrava, "Astronomy in Prague: from the past to the present," *Proc. Int. Astron. Union*, vol. 2, no. 14, Dec. 2007.

[23] "CATHOLIC ENCYCLOPEDIA: Astronomy." [Online]. Available: http://www.newadvent.org/cathen/02025a.htm. [Accessed: 06-Feb-2014].

[24] D. Arditti, *Setting-Up a Small Observatory: From Concept to Construction*, 1º. Springer, 2008.

[25] N. G. Hockstein, C. G. Gourin, R. a. Faust, and D. J. Terris, "A history of robots: From science fiction to surgical robotics," *J. Robot. Surg.*, vol. 1, no. 2, pp. 113–118, 2007.

[26] V. Kumar, "50 Years of Robotics [From the Guest Editors," *IEEE Robot. Autom. Mag.*, vol. 17, no. 3, pp. 8–8, Sep. 2010.

[27] *The Robosapien Companion*. Berkeley, CA: Apress, 2005.

[28] "Home - World Robotics 2013." [Online]. Available: http://www.worldrobotics.org/index.php?id=home. [Accessed: 02-Mar-2014].

[29] "Robot Terms and Definitions - Robotic Industries Association." [Online]. Available: http://www.robotics.org/product-catalog-detail.cfm/Robotic-Industries-Association/Robot-Terms-and-Definitions/productid/2953. [Accessed: 07-Feb-2014].

[30] U. Kartoun, *Virtual Reality Telerobotic System*. e-ENGDET 2004 4th International Conference on e-Engineering and Digital Enterprise Technology, 2004.

[31] A. Rovetta, A. K. Bejczy, and R. Sala, “Telerobotic surgery: applications on human patients and training with virtual reality.,” *Stud. Health Technol. Inform.*, vol. 39, pp. 508–17, Jan. 1997.

[32] “CODELCO - Corporación Nacional del Cobre , Chile Minera Gaby consolida la operación de la mayor flota de camiones autónomos del mundo.” [Online]. Available: http://www.codelco.com/minera-gaby-consolida-la-operacion-de-la-mayor-flota-de-camiones-autonomos-del-mundo/prontus_codelco/2012-04-18/192119.html. [Accessed: 12-Feb-2014].

[33] H. Hirukawa, T. Matsui, H. Onda, K. Takase, Y. Ishiwata, and K. Konaka, “Prototypes of teleoperation systems via a standard protocol with a standard human interface,” in *Proceedings of International Conference on Robotics and Automation*, 1997, vol. 2, pp. 1028–1033.

[34] O. Michel, P. Saucy, and F. Mondada, “‘KhepOnTheWeb’: An experimental demonstrator in telerobotics and virtual reality,” in *Proceedings. International Conference on Virtual Systems and MultiMedia VSMM ’97 (Cat. No.97TB100182)*, 1997, pp. 90–98.

[35] K. Goldberg, B. Chen, R. Solomon, S. Bui, B. Farzin, J. Heitler, D. Poon, and G. Smith, “Collaborative teleoperation via the Internet,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, 2000, vol. 2, pp. 2019–2024.

[36] Xie Xiaohui, Du Zhijiang, and Sun Lining, “The design and implementation of real-time internet-based telerobotics,” in *IEEE International Conference on Robotics, Intelligent Systems and Signal Processing, 2003. Proceedings. 2003*, 2003, vol. 2, pp. 815–819.

[37] P. Fiorini and R. Oboe, “Internet-based telerobotics: problems and approaches,” in *1997 8th International Conference on Advanced Robotics. Proceedings. ICAR’97*, 1997, pp. 765–770.

[38] T. Kusu, Y. Ito, T. Kida, T. Shimada, M. Takahashi, Y. Nomoto, Y. Tsuchiya, M. Narita, and Y. Kato, “A Virtual Campus Tour Service Using Remote Control Robots on Robot Service Network Protocol,” in *2013 27th International Conference on Advanced Information Networking and Applications Workshops*, 2013, pp. 959–964.

[39] C. Preeda and G. Gueng Hwang, “Implementation of internet based telemicromanipulation with delay compensation,” in *2008 10th IEEE International Workshop on Advanced Motion Control*, 2008, pp. 312–317.

[40] T. Szymanski and D. Gilbert, “Provisioning mission-critical telerobotic control systems over internet backbone networks with essentially-perfect QoS,” *IEEE J. Sel. Areas Commun.*, vol. 28, no. 5, pp. 630–643, Jun. 2010.

[41] I. Elhajj, Y. Hasegawa, and T. Fukuda, “Supermedia-enhanced internet-based telerobotics,”

*Proc. IEEE*, vol. 91, no. 3, pp. 396–421, Mar. 2003.

[42] Jahng-Hyon Park and Joonyoung Park, “Real time bilateral control for Internet based telerobotic system,” in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003) (Cat. No.03CH37453)*, 2003, vol. 2, pp. 1106–1110.

[43] J. J. Craig, *Introduction to Robotics: Mechanics and Control, 3rd, Craig*. .

[44] S. Srinivasan, “Design patterns in object-oriented frameworks,” *Computer (Long. Beach. Calif).*, vol. 32, no. 2, pp. 24–32, 1999.

[45] R. E. Johnson and B. Foote, “Designing Reusable Classes,” *J. Object-Oriented Softw.*, vol. 1, no. 2, pp. 22–35, 1988.

[46] M. E. Fayad, D. C. Schmidt, and R. E. Johnson, *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, 1st ed. Wiley, 1999.

[47] “Object Management Group.” [Online]. Available: http://www.omg.org/. [Accessed: 24-Feb-2014].

[48] ROS, “ROS/Concepts - ROS Wiki,” 2014. [Online]. Available: http://wiki.ros.org/ROS/Concepts. [Accessed: 11-Mar-2014].

[49] ROS, “ROS.org | Powering the world’s robots.” [Online]. Available: http://www.ros.org/. [Accessed: 11-Mar-2014].

[50] “Player Project.” [Online]. Available: http://playerstage.sourceforge.net/. [Accessed: 11-Mar-2014].

[51] “YARP - Yet Another Robot Platform.” [Online]. Available: http://eris.liralab.it/yarp/. [Accessed: 11-Mar-2014].

[52] “The Orocos Project | Smarter control in robotics & automation!” [Online]. Available: http://www.orocos.org/. [Accessed: 11-Mar-2014].

[53] “CARMEN.” [Online]. Available: http://carmen.sourceforge.net/. [Accessed: 11-Mar-2014].

[54] “Orca Robotics.” [Online]. Available: http://orca-robotics.sourceforge.net/. [Accessed: 11-Mar-2014].

[55] “MOOS : Main - Home Page browse.” [Online]. Available: http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php. [Accessed: 11-Mar-2014].

[56] “Microsoft Robotic Studio.” [Online]. Available: http://www.microsoft.com/en-us/download/details.aspx?id=29081. [Accessed: 11-Mar-2014].

[57] A. Wallander, “Using NTT as prototype for VLT: the new NTT control system,” in *Optical Telescopes of Today and Tomorrow*, 1997, pp. 1005–1011.

[58] E. Allaert and G. Raffi, “Preparation for VLT software commissioning at Paranal,” in *Optical Science, Engineering and Instrumentation ’97*, 1997, pp. 20–27.

[59] G. Raffi and K. Wirenstrand, “VLT control software in its final test phase,” in *Optical Telescopes of Today and Tomorrow*, 1997, pp. 996–1004.

[60] E. Pozna, G. Zins, P. Santin, and S. Beard, “A common framework for the observation software of astronomical instruments at ESO,” in *SPIE Astronomical Telescopes + Instrumentation*, 2008, p. 70190Q–70190Q–12.

[61] E. Pozna, “Evolution of the top level control software of astronomical instruments at ESO,” in *SPIE Astronomical Telescopes + Instrumentation*, 2012, p. 845107.

[62] M. J. Kiekebusch, G. Chiozzi, J. Knudstrup, D. Popovic, and G. Zins, “Evolution of the VLT instrument control system toward industry standards,” in *SPIE Astronomical Telescopes + Instrumentation*, 2010, p. 77400T–77400T–12.

[63] W. Pessemier, G. Raskin, S. Prins, P. Saey, F. Merges, J. P. Padilla, H. Van Winckel, and C. Waelkens, “Towards a new Mercator Observatory Control System,” in *SPIE Astronomical Telescopes + Instrumentation*, 2010, p. 77403B–77403B–10.

[64] D. Popovic, R. Brast, N. Di Lieto, M. Kiekebusch, J. Knudstrup, and C. Lucuix, “Motion control solution for new PLC-based standard development platform for VLT instrument control systems,” 2014, p. 915209.

[65] G. Chiozzi, B. Gustafsson, B. Jeram, M. Plesko, M. Sekoranja, G. Tkacik, and K. Zagar, “CORBA-based Common Software for the ALMA project,” in *Astronomical Telescopes and Instrumentation*, 2002, pp. 43–54.

[66] J. Schwarz, A. Farris, and H. Sommer, “The ALMA software architecture,” in *Astronomical Telescopes and Instrumentation*, 2004, pp. 190–204.

[67] G. Chiozzi, B. Jeram, H. Sommer, A. Caproni, M. Plesko, M. Sekoranja, K. Zagar, D. W. Fugate, P. Di Marcantonio, and R. Cirami, “The ALMA common software: a developer friendly CORBA-based framework,” in *Astronomical Telescopes and Instrumentation*, 2004, pp. 205–218.

[68] E. Jeschke, B. Bon, T. Inagaki, and S. Streeper, “A framework for the Subaru Telescope observation control system based on the command design pattern,” in *SPIE Astronomical Telescopes + Instrumentation*, 2008, p. 70190O–70190O–12.

[69] B. Grigsby, K. Chloros, J. Gates, W. T. S. Deich, E. Gates, and R. Kibrick, “Remote observing with the Nickel Telescope at Lick Observatory,” in *SPIE Astronomical Telescopes + Instrumentation*, 2008, pp. 701627-701627–12.

[70] R. I. Kibrick, G. D. Wirth, E. L. Gates, B. J. Grigsby, W. T. S. Deich, K. Lanclos, and S. L.

Allen, “A shared approach to supporting remote observing for multiple observatories,” in *SPIE Astronomical Telescopes + Instrumentation*, 2010, pp. 773712-773712–12.

[71] P. Kubánek, A. J. Castro-Tirado, A. de Ugarte Postigo, R. Cunniffe, M. Prouza, J. Štrobl, H. van Heerden, J. Gorosabel, R. Hudec, P. Yock, W. H. Allen, I. Bond, G. Christie, S. Guziy, L. Hanlon, M. Jelínek, S. Meehan, C. Polášek, V. Reglero, and P. Vitale, “Operating a global network of autonomous observatories,” in *SPIE Astronomical Telescopes + Instrumentation*, 2010, p. 77400U–77400U–12.

[72] A. F. Żarnecki, L. W. Piotrowski, L. Mankiewicz, and S. Małek, “Analysis framework for GLORIA,” in *Photonics Applications in Astronomy, Communications, Industry, and High-Energy Physics Experiments 2012*, 2012, p. 845408.

[73] A. Ferayorni, “Instrument control software for the Visible Broadband Imager using ATST common services framework and base,” in *SPIE Astronomical Telescopes + Instrumentation*, 2012, p. 845113.

[74] M. Weber, T. Granzer, and K. G. Strassmeier, “The STELLA robotic observatory on Tenerife,” in *SPIE Astronomical Telescopes + Instrumentation*, 2012, p. 84510K.

[75] J. Dong, J. Wang, G. Jin, X. Deng, and H. Yuan, “Research of Application Layer for Large Astronomical Telescope Observatory Control System Framework,” in *2009 International Conference on Computational Intelligence and Software Engineering*, 2009, pp. 1–4.

[76] “ASCOM - Standards for Astronomy.” [Online]. Available: http://ascom-standards.org/. [Accessed: 14-Dec-2014].

[77] “TheSkyX Professional Edition - Software Bisque.” [Online]. Available: http://www.bisque.com/sc/pages/TheSkyX-Professional-Edition.aspx. [Accessed: 14-Dec-2014].

[78] “INDI Library.” [Online]. Available: http://www.indilib.org/. [Accessed: 14-Dec-2014].

[79] “Stellarium.” [Online]. Available: http://www.stellarium.org/. [Accessed: 14-Dec-2014].

[80] “RTS2: open source standard and package for autonomous observatory.” [Online]. Available: http://www.rts2.org/. [Accessed: 14-Dec-2014].

[81] P. Wu and C. Luo, “ASCOM based research on the universal control protocol of telescope,” in *7th International Symposium on Advanced Optical Manufacturing and Testing Technologies (AOMATT 2014)*, 2014, p. 92830Q.

[82] “RTS2.” [Online]. Available: http://spie.org/Publications/Proceedings/Paper/10.1117/12.672045. [Accessed: 14-Dec-2014].

[83] “The RTS2 protocol.” [Online]. Available: http://spie.org/Publications/Proceedings/Paper/10.1117/12.788623. [Accessed: 14-Dec-2014].

[84] “RTS2: meta-queues scheduling and its realisation for FLWO 1.2m telescope.” [Online]. Available: http://spie.org/Publications/Proceedings/Paper/10.1117/12.926839. [Accessed: 14-Dec-2014].

[85] P. Kubánek, “RTS2 — Remote Telescope System, 2nd Version,” in *AIP Conference Proceedings*, 2004, vol. 727, no. 727, pp. 753–756.

[86] “Software solution for autonomous observations with H2RG detectors and SIDECAR ASICs for the RATIR camera.” [Online]. Available: http://spie.org/Publications/Proceedings/Paper/10.1117/12.925817. [Accessed: 15-Dec-2014].

[87] “Design of modular C++ observatory control system: from observatories to laboratories and back.” [Online]. Available: http://spie.org/Publications/Proceedings/Paper/10.1117/12.857258. [Accessed: 15-Dec-2014].

[88] Raspberry PI, “FAQs | Raspberry Pi.” [Online]. Available: http://www.raspberrypi.org/help/faqs/. [Accessed: 12-Jun-2014].

[89] P. Abrahamsson, S. Helmer, N. Phaphoom, L. Nicolodi, N. Preda, L. Miori, M. Angriman, J. Rikkila, X. Wang, K. Hamily, and S. Bugoloni, “Affordable and Energy-Efficient Cloud Computing Clusters: The Bolzano Raspberry Pi Cloud Cluster Experiment,” in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science*, 2013, vol. 2, pp. 170–175.

[90] “WeeWX: Linux weather software.” [Online]. Available: http://www.weewx.com/. [Accessed: 05-Oct-2015].